Adaptive Mensch-Maschine Interaktion für mobile Agenten

Vom Fachbereich Informatik (FB 20)
der Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

von

Dipl.-Ing. Eric Blechschmitt

geboren in Darmstadt

Referent:	Prof. Dr.-Ing. Dr. h.c. Dr. E. h. José L. Encarnação Technische Universität Darmstadt
Koreferent:	Prof. Dr.-Ing. Thomas Kirste Universität Rostock
Tag der Einreichung:	1.6.2005
Tag der mündlichen Prüfung:	18.7.2005

Darmstädter Dissertation

D17

Darmstadt, 2005

Danksagung

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter in der Abteilung „Animation & Bildkommunikation“ am Fraunhofer-Institut für Graphische Datenverarbeitung in Darmstadt.

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Dr. h.c. Dr. E. h. José L. Encarnação für die Förderung dieses interessanten Themas und die Betreuung meiner Arbeit. Herrn Prof. Dr.-Ing. Thomas Kirste danke ich für die Übernahme des Koreferats.

Einen ebenso wichtigen Anteil an meiner Arbeit haben Matthias Finke und Dr. Norbert Braun, denn ohne sie hätte ich mich wohl nie aufgerafft diese Arbeit anzufangen, geschweige denn zu beenden. Ein großer Dank gilt auch Axel Brümmer und Kai Richter für die vielen anregenden Gespräche, die mir weitere Ansichten auf mein Thema gaben. Detlef Krömker, Reinhard Klein, Volker Luckas und Jörn Kohlhammer danke ich, weil sie als Abteilungsleiter die Basis für meine Arbeit gaben.

Lisa und Anna danke ich für die Zeit, die sie mir für diese Arbeit geschenkt haben. Außerdem für ihre Mühe, ihr Verständnis und für so manche nötige Ablenkung.

Meinen Eltern möchte ich danken, dass sie mir diesen Weg ermöglicht haben.

Kurzfassung

Der Einsatz von mobiler Telekommunikation und Datenverarbeitung ist inzwischen in allen Lebensbereichen selbstverständlich geworden. Die Entwicklung ist jedoch noch nicht an ihrem Ende angelangt, sondern sie schreitet weiter fort und erlaubt die Herstellung von immer kleineren und in zunehmendem Maße leistungsfähigeren Endgeräten. Die Möglichkeit, das Internet nun auch drahtlos nutzen zu können, hat der mobilen Datenverarbeitung eine neue Qualität verliehen. Internet-Dienste sind nicht mehr an den Gebrauch traditioneller Desktop-PCs gebunden, sondern können nun auch über mobile Endgeräte verschiedenster Ausprägung genutzt werden. Laptop-PCs, Personal Data Assistants und sogar Mobiltelefone kommen zum Einsatz. Durch die Vernetzung dieser Geräte entsteht eine Plattform für verteilte Anwendungen. Als Basis einer Realisierung von verteilten und mobilen Anwendungen empfehlen sich u.A. mobile Agentensysteme. Gegenstand dieser Arbeit ist die Auswirkung der technologischen Entwicklung auf agentenbasierte Netz-Anwendungen. Im Fokus steht die dynamische Adaption von Benutzungsschnittstellen an mobile und stationäre Endgeräte.

Diese Arbeit stellt ein Modell vor, das zur Entwicklung von mobilen Agentenanwendungen herangezogen werden kann, die mobile und stationäre Endgeräte integrieren. Das Modell definiert die Architektur eines agentenbasierten mobilen User Interface Management Systems anhand einer Dialogkomponente, einer Adapterkomponente und mehrerer Präsentations-komponenten, die koordiniert zusammenarbeiten. Die Synchronisation der Interaktionsdaten ist sowohl in den Präsentationsmodulen, als auch in der Dialogkomponente vorgesehen. Die Dialogkomponente stützt sich auf ein Interaktions-Objektmodell, dessen Elemente sich gegenseitig synchronisieren. Das Objektmodell wird aus einer abstrakt kodierten Dialogsprache gewonnen.

Um die Tragfähigkeit des Konzepts nachzuweisen und die Überlegenheit des Ansatzes gegenüber dem Stand der Technik zu zeigen, wurde auf der Basis der entworfenen Architektur ein prototypisches System realisiert. Das System ermöglicht 1. die Kodierung abstrakter Benutzungsoberflächen, 2. die Adaption der abstrakten Benutzungsoberflächen an verschiedene stationäre und mobile Endgeräte und 3. die kontinuierliche Ausführung von Dialogen über die Grenzen verschiedener Endgeräte hinweg. Die Arbeit schließt mit einem Vergleich der entwickelten Lösung mit existierenden Systemen.

Abstract

Mobile communication technologies and mobile computing devices have evolved into a common part of our everyday life. Advances in the production of electronic components enable the manufacturing of smaller and increasingly more powerful devices. The additional option to access the internet wireless emphasises the significance to mobile computing devices. Internet services are no longer bounded to the use of desktop PCs but more and more by means of a broad spectrum of mobile devices such as Personal Data Assistants or mobile telephones.

By the integration of multiple devices in a network a potential computing platform arises, which enables mobile and distributed application programs. Mobile agent platforms matches particular requirements of mobile application programs and can be used as a runtime environment. This work examines the impact of this development on network based agent applications and focuses the dynamic adaptation of user interfaces to mobile and stationary devices.

The dissertation presents a model for the development of mobile agent applications which integrate various mobile and stationary devices. The model defines the architecture of an agent based and mobile User Interface Management System consisting of a dialog component, an adaptation component and multiple synchronized presentation components. The synchronization is performed on the presentation-layer and also on the dialog layer. The dialog component uses an interaction object model, where the elements do synchronize each other. The object model is generated from an abstract dialog language.

On the base of the concepts of this work a concrete implementation has been realized to verify the advantages of the approach in comparison with the state of the art. The system enables 1. to encode User Interfaces in an abstract manner, 2. to adapt once encoded User Interfaces to various devices and 3. continuous execution of dialogs among different devices.

The work closes with an evaluation and comparison of different existing user interface transcoding systems with respect to software engineering concerns.

Inhaltsverzeichnis

1 Einleitung

Gegenstand dieser Arbeit ist die Modellierung einer Infrastruktur für mobile Agentensysteme, die Benutzungsoberflächen dynamisch an stationäre und mobile Endgeräte anpasst. Damit wird ein Konzept für die Realisierung von mobilen agentenbasierten Informationsdiensten gegeben. Die Arbeit wurde durch die These motiviert, dass ein abstraktes Kodierungsschema möglich ist, mit dem sich Benutzungsoberflächen *einmal* kodieren und *vielfach* auf *unterschiedlichen* Endgeräten darstellen lassen. Die gefundene Lösung konzentriert sich auf Aspekte der Dialogkontrolle, um die Interaktion auf kleinen Geräten mit eingeschränkten Interaktionsmöglichkeiten für den Benutzer komfortabler zu gestalten. Zudem erlaubt die modulare Architektur die Komposition von Benutzungsoberflächen aus verschiedenen Modalitäten.

1.1 Hintergrund

Einer Meldung der deutschen Presseagentur (dpa) zufolge überstieg im ersten Quartal des Jahres 2002 die Anzahl der weltweit eingesetzten Mobiltelefone erstmals die Anzahl der Festnetzanschlüsse [Heis02]. Diese Meldung dokumentiert anschaulich, dass der Einsatz von mobiler Telekommunikation inzwischen alltäglich ist. Als Folge dieser Alltäglichkeit zeigen sich schon neue Verhaltensmuster bei der „Handy-Generation“: Es wird kurzfristiger geplant, Verabredungen werden spontan getroffen und man verlässt sich auf die Technik. Mobiltelefone sind heute sowohl im geschäftlichen als auch privaten Bereich selbstverständlich.

Jahr	**1999**	**2000**	**2001**	**2002**	**2003**	**2004**
Desktop-PC	43,1 %	45,6 %	51,6 %	54,1 %	57,8 %	58,7 %
Laptop-PC	4,9 %	5,5 %	6,1 %	7,9 %	9,9 %	13,3 %

Tabelle 1: Ausstattungsgrad deutscher Privathaushalte mit Desktop- und Laptop-PC [Dest05]

Im Gebrauch befinden sich nun zunehmend Geräte, die Kommunikation und Datenverarbeitung vereinen und den Zugang zu Kommunikations- und Informationsdiensten weitgehend ortsflexibel ermöglichen. Mobiltelefone sowie tragbare und kleinste EDV-Geräte für unterwegs, die Telekommunikation und Datenverarbeitung integrieren, sind zu erschwinglichen Massenprodukten gereift. Die seit Jahrzehnten anhaltende Miniaturisierung der elektronischen Datenverarbeitung [Moor65] hat durch die mobile Nutzbarkeit eine neue Qualität erreicht.

Mobiles EDV-Gerät wird zunehmend nachgefragt. Stand vor wenigen Jahren noch die Leistungsfähigkeit von Computern im Vordergrund, wirken nun andere Faktoren kaufentscheidend: Niedriger Platzbedarf sowie die Möglichkeit mobil zu arbeiten sind wichtige Kriterien. Die Zahlen des statistischen Bundesamtes [Dest05] in Tabelle 1 zeigen, dass der Anteil der privat genutzten Laptop-PCs gegenwärtig noch weit hinter dem Anteil der stationär genutzten Desktop-PCs zurück liegt. Jedoch stört die geringere Leistungsfähigkeit und der höhere Preis der mobilen Laptop-PCs die Privatanwender offenbar immer weniger. Der direkte Vergleich des *Wachstums* des Ausstattungsgrades in Abbildung 1 offenbart dementsprechend deutliche Zuwachsraten zugunsten der Laptop-PCs.

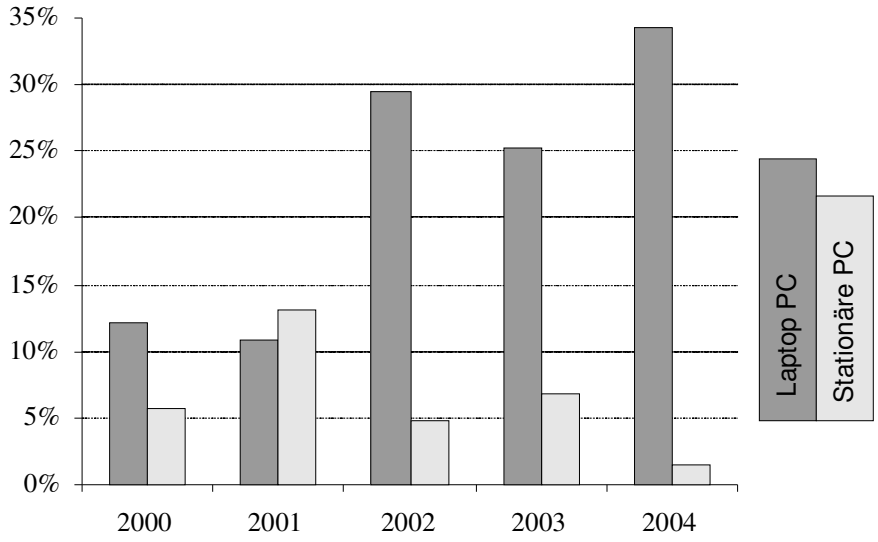

Abbildung 1: Wachstum des Ausstattungsgrades der deutschen Privathaushalte mit PCs

Stationäre und mobile EDV-Geräte sind zunehmend für den Zugriff auf Netzwerke vorbereitet. Dabei hat sich die Netzwerktechnik in den vergangenen Jahren in Bezug auf die nutzbare Bandbreite und durch Standards zur drahtlosen Datenübertragung weiterentwickelt. Standards wie Bluetooth, WLAN[1], GSM[2] und UMTS[3] erlauben einen noch flexibleren Zugriff auf Datendienste. In funkbasierten Netzwerken entfällt die Notwendigkeit einer starren Anschlussleitung. Verbindungen können *ad hoc* hergestellt und getrennt werden. Darüber hinaus können sich die Teilnehmer eines Funknetzwerks innerhalb einer bestimmten Reichweite bewegen.

[1] WLAN nach 802.11b hat innerhalb von Gebäuden eine Reichweite von 10 bis 100 Metern bei etwa 10Mps [IEEE99].

[2] GSM = Global System for Mobile communication.

[3] UMTS = Universal Mobile Telecommunications System

Diese Entwicklung wurde durch die anhaltende Zunahme[4] der Bauteiledichte auf den Chipflächen ermöglicht. Dadurch können immer mehr verschiedenartige Funktionen in einem Gerät integriert werden. Neben ihrer primären Funktion erfüllen heutige Geräte zunehmend sekundäre Aufgaben. Aktuelle Mobiltelefone werden neben ihren Telekommunikationsaufgaben auch als elektronisches Adressbuch, Notizblock, Spielcomputer, zum Anfertigen von Fotos, zum Abspielen von Musik, als Datenspeicher, etc. genutzt. Ein solcher Mobilfunk-PDA ist in Abbildung 2 dargestellt. Diese „Verschmelzung" vormals verschiedener Anwendungsfelder bedingt die Kooperation der Branchen Telekommunikation, Medien und Informationstechnologien und wird auch als „Konvergenz der Märkte" bezeichnet.

Abbildung 2: Ein Mobilfunk-PDA[5] vereint Mobiltelefon und PDA [Heis03]

Überall und zu jeder Zeit können Daten über hohe Bandbreiten auf mobile Endgeräte übermittelt und vor Ort verarbeitet werden. Das schafft Potenziale für Dienstleistungen, die besser auf den Benutzer, seinen Aufenthaltsort und die Zeit abgestimmt werden können, als es bisher möglich war. Wohin geht diese Entwicklung? Encarnação präsentiert in [Enca03] eine Technologie Roadmap, die in der Verwirklichung von *intelligenten Umgebungen* und dem *Perceptual Computing* mündet. Die im Hintergrund kommunizierenden Geräte werden dann Teile der Umwelt darstellen, die durch eine Vielzahl inhomogener Geräte gekennzeichnet sein wird.

[4] Gordon Moore prognostizierte bereits 1965 die Verdopplung der Bauteiledichte ca. alle 2 Jahre [Moor65].

[5] PDAs (Personal Data Assistant) sind Kleincomputer, die z.B. als elektronische Terminkalender verwendet werden.

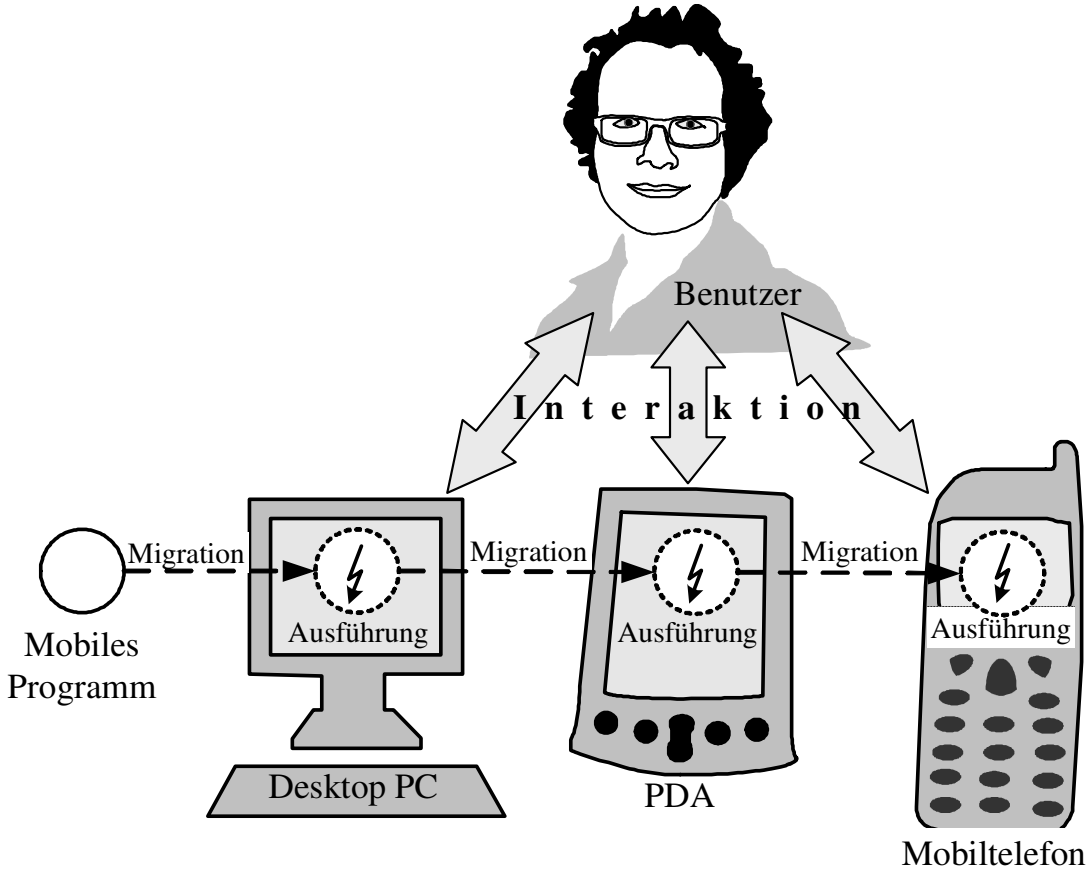

Abbildung 3: Ein mobiles Anwendungsprogramm nutzt verschiedene Endgeräte als Plattform

Dieses Netzwerk aus kommunizierenden Geräten bildet eine ideale Plattform für neuartige Anwendungsprogramme [SVBL01][Duca01]. Solche mobilen Anwendungsprogramme können sich, wie in Abbildung 3 illustriert, von Gerät zu Gerät bewegen und Aufgaben nach und nach bearbeiten. Prototypische Systeme zur Realisierung solcher mobilen Anwendungen wurden bereits mehrfach vorgestellt. u.A. durch Bharat und Cardelli [BC95] und im Rahmen von Projekten zur Gestaltung zukünftiger Büroarbeitsplätze, wie etwa *OfficePlus* [BS01][Blec99].

Derartige mobile Anwendungsprogramme sollen im Rahmen dieser Arbeit nach Definition 1 als *mobile Agenten* bezeichnet werden.

Definition 1: *Mobile Agenten* sind Softwareprogramme, die während ihrer Ausführung auf ein anderes Rechnersystem migrieren und die Ausführung auf dem Zielsystem fortsetzen können.

1.2 Motivation

Auch für die nähere Zukunft kann von wachsender Performanz bei elektronischen Geräten und von steigenden Übertragungsbandbreiten ausgegangen werden [Moor65][GGPY89]. Die zunehmende Verbreitung von stationärer und mobiler EDV und ihre Vernetzung ebnet den Weg zu deren Nutzung als gemeinsame Plattform für mobile Agenten nach Definition 1. Die Prüfung und Verifikation der sich ergebenden Potenziale für mobile Agenten ist ein Anliegen der Informationsgesellschaft, da sich Konsequenzen für das Arbeiten und Wirtschaften ergeben [Reich02]. Nach Encarnação ist die *„...Entwicklung und die Integration fortschrittlicher und innovativer Basiswerkzeuge und Interaktionsschnittstellen erforderlich."* [Enca99]. Somit ist die Modellierung geeigneter Architekturen und die Realisierung von Prototypen notwendig, anhand derer Validierungen möglich sind.

1.2.1 Anwendung

Anwendungsfelder für mobile Agenten finden sich besonders im wirtschaftlich bedeutenden Bürobereich, der von Arbeitsteilung, Koordination und Internationalisierung geprägt ist. Das *Mobile Office* wird durch mobile Agenten ergänzt und soll nun Funktionen realisieren, die bisher fest installierten Arbeitsplätzen vorbehalten waren. Als Beispiel ist im Folgenden das Szenario der mobilen Terminvereinbarung mit Hilfe eines *Terminagenten* beschrieben.

Der Terminagent unterstützt den Benutzer bei der Vereinbarung eines gemeinsamen Termins mit mehreren Personen. Eine Terminvereinbarung mit vielen Teilnehmern stellt oft einen erheblichen Aufwand dar. Üblicherweise kommuniziert der Benutzer eine Liste mit Terminvorschlägen an die Teilnehmer. Dies erfolgt in der Hoffnung, dass es mindestens einen Termin gibt, den alle Teilnehmer bestätigen können. Nach Erhalt aller Antworten prüft der Benutzer die Antworten. Wird kein gemeinsamer Termin gefunden, wiederholt sich die gesamte Prozedur. Der Terminagent hilft nun dem Benutzer in dieser Situation und erledigt diese iterative Aufgabe weitgehend selbstständig. Dazu entnimmt der Terminagent freie Termine des Benutzers aus dessen elektronischen Kalender und migriert auf die Endgeräte der Partner. Dort nimmt er Kontakt mit den Teilnehmern – oder mit deren Terminagenten - auf, um einen freien Termin zu verhandeln. Ist ein Termin gefunden, der den Bedürfnissen aller Teilnehmer entspricht, kehrt der Agent wieder zu dem Endgerät seines Benutzers zurück und meldet das Ergebnis.

Dieses Szenario verdeutlicht die vorteilhafte Verwendung der Agententechnologie und es zeigt einen Paradigmenwechsel auf, der die Sicht des Menschen auf die Software verändern wird: Ein *Anwendungsprogramm* wird *benutzt*, während ein *Softwareagent* als *Assistent* und seine Beauftragung als *Delegation* wahrgenommen wird. Die Verwendung von mobilen Agenten setzt das Vertrauen des Auftraggebers und des Empfängers in den Agenten voraus [Norm94]. Entscheidend wirken die Freiheitsgrade der Agenten. Ein *mobiler Agent* kann sich im Netzwerk bewegen und unabhängig vom Benutzer auf verschiedenen Computern agieren. Als *autonomer*

Agent[6] kann er selbständig Entscheidungen treffen und ist nicht auf das Eingreifen des Benutzers angewiesen [Maes94]. Dies führt zu ganz neuen Herausforderungen in der Mensch-Maschine Interaktion [Kirst02]. So muss etwa den Assistenz- und Delegationsfunktionen durch geeignete Metaphern Ausdruck verliehen werden, da sie sonst nicht als solche erkannt werden [AR00].

Die Möglichkeiten mobiler und autonomer Agenten stellen nicht nur Lösungen dar, sondern sie verursachen wiederum neue technische Probleme, die gelöst werden müssen. Damit die Vision in alltagstaugliche Prototypen umgesetzt werden kann, müssen Softwareagenten mit dem Benutzer *mobil interagieren* können. Das bedeutet entsprechend dem Ziel von *Mobile Computing*: Zu jeder Zeit, an jedem Ort und damit auch *auf jedem Endgerät*.

1.2.2 Bedarf

Die Entwicklung von Diensten auf der Basis von mobilen Agenten unterscheidet sich grundsätzlich von der Anwendungsentwicklung für stationäre Systeme. Stationär arbeitende Programme sind für eine bestimmte Geräteklasse ausgelegt und werden im Rahmen einer Installationsprozedur an ein System angepasst und personalisiert. Dadurch können Rahmenbedingungen, wie Betriebssystem, Displaygröße, Ausführung der Tastatur und Benutzervorgaben Berücksichtigung finden.

Mobile Agenten bewegen sich hingegen in einem Netzwerk aus verschiedenen Endgeräten. Sie können im Netz lokalisierte Dienste für die Lösung einer Aufgabe einsetzen. Dieses Netzwerk befindet sich nicht in einem statischen Zustand, sondern ist von einem sich wandelnden Dienstangebot gekennzeichnet. Um die sich ergebenden Konsequenzen für den Lebenszyklus eines Agenten zu berücksichtigen, kann das Applikationsmodell nach [BFLM+92] herangezogen werden. Es unterteilt den Lebenszyklus einer mobilen Applikation in *Entwicklung* (Design-time), *Ladevorgang* (Load-time) und *Laufzeit* (Runtime).

Die *Entwicklung* einer mobilen Applikation sollte demnach

- aufgabenbezogen und geräteunabhängig erfolgen,
- trennen zwischen Anwendungslogik und Benutzungsschnittstelle und
- mögliche Umgebungsdienste berücksichtigen.

Der *Ladevorgang* ist verknüpft mit

- der Identifikation von Diensten im Netzwerk und der Möglichkeit zum Zugriff,
- der Zugreifbarkeit der Ressourcen des Endgeräts für die mobilen Applikationen und
- der Darstellung von Benutzungsoberflächen und deren Anpassung an das Gerät.

[6] Vgl. Abschnitt 3.5

In der *Laufzeit* muss eine mobile Applikation

- auf Änderungen in der Umgebung reagieren können,
- mit dem Wechsel des Verbindungsstatus umgehen können und
- einen Programmzustand gezielt speichern und wieder herstellen können.

Bei näherer Betrachtung zeigt sich, dass einige der o.g. Forderungen in mobilen Agentensystemen bereits realisiert sind. Aspekte wie der *Ladevorgang* und die *Laufzeit* typische Problemstellungen mobiler Agentensysteme. Darüber hinaus ergeben sich folgende Vorteile:

- Mit Agenten kann die *Erkennung von Diensten und deren Nutzung* sehr einfach realisiert werden. (vgl. Directory Facilitator in Abschnitt 3.5.3).
- Endgeräte können durch die Agentenplattform als Laufzeitumgebung standardisiert werden (*Middleware*[7]). Damit ist bei Programmablauf die *Geräteunabhängigkeit* gewährleistet.
- Agenten können *aufgabenbezogen* realisiert werden (vgl. Terminagent, Abschnitt 1.2.1).
- Agenten können *Änderungen in ihrer Umgebung* erkennen (vgl. Definition in Abschnitt 3.5).
- Mobile Agenten können bei einer Migration den *Programmzustand gezielt speichern und wieder herstellen*.
- Zur der *Trennung der Anwendungslogik von der Benutzungsschnittstelle* kann ein Dienst für die Mensch-Maschine Interaktion in die Agentenplattform bzw. in eine spezielle Instanz eines Agenten (etwa als *User Interface Agent*) integriert werden.

Offenbar eignen sich mobile Agentensysteme gut für die Implementation der mobilen Applikationen nach [BFLM+92]. Jedoch sind folgende offenen Forderungen zu erfüllen:

- Die Aufgabenbezogenheit der Interaktionsdaten.
- Die Speicherung und Wiederherstellung des Zustands der Benutzungsoberfläche.
- Die Anpassung der Benutzungsoberfläche an das Endgerät.

Dies stellt eine erhebliche Herausforderung dar, da

- die Interaktionsschnittstellen der Endgeräte sehr unterschiedlich ausgeprägt sind und
- der Markt für mobile Engeräte von sehr kurzen Produktzyklen geprägt ist.

Im Rahmen dieser Arbeit wird der Frage nachgegangen, wie die Interaktionsmöglichkeiten mobiler Endgeräte durch Softwareagenten besser ausgenutzt werden können, ohne den Entwicklungsaufwand unangemessen zu erhöhen. Somit soll die Realisierung von mobilen Softwareanwendungen vereinfacht werden, die mit mobilen Endgeräten arbeiten. Das Ziel soll durch die Konzeption einer Architektur erreicht werden, welche die Mensch-Maschine Interaktion gezielt unterstützt und die o.g. noch offenen Forderungen nach [BFLM+92] erfüllt.

[7] So etwa die Java Virtual Machine (JVM) eine standardisierte Laufzeitumgebung zur Ausführung von Javakode ist.

1.3 Einordnung

Das Thema dieser Arbeit, die *Adaptive Mensch-Maschine Interaktion für mobile Agenten,* wird nun in einen größeren Zusammenhang mit Forschungsgebieten und Fragestellungen der Informatik und mit der Unterdisziplin der *Graphischen Datenverarbeitung* gestellt [FVFH95].

Eine zentrale Aufgabenstellung der Computergraphik ist die *Präsentation* von Informationen und ihre *Visualisierung*. Die Präsentation von Informationen geschieht auch bei der Mensch-Maschine Interaktion. Da die Interaktion das Bindeglied zwischen Mensch und Maschine darstellt, kann sie als potentieller Engpass zum Schwachpunkt einer Anwendung werden. Die Informationsvisualisierung beschäftigt sich vorrangig mit der Ausnutzung des menschlichen Wahrnehmungssystems, um Daten effizient explorieren zu können [SM04]. Im vorliegenden Fall werden Präsentationstechniken für Interaktionsdaten behandelt, die verschiedene Engeräte adaptieren.

Der Bereich der Mensch-Maschine Interaktion beschäftigt sich u.A. mit der Frage der Anpassbarkeit von Interaktionsdaten an verschiedene Bedienschnittstellen und –metaphern. Auch ihre Teildisziplin der *multimodalen Interaktion* ist von dieser Arbeit berührt. Ein System, das sich an unterschiedliche Konfigurationen verschiedener Interaktionsmedien anpasst, muss die Fähigkeit besitzen, diese Medien zu kontrollieren und zu synchronisieren. Die Auswahl, Anpassung und Synchronisation von verschiedenen Interaktionskanälen ist eine zentrale Frage der multimodalen Interaktionsschnittstellen [Bolt80][Ovia99].

Weiterhin umfasst das Anwendungsspektrum des Lösungsansatzes Methoden, die *kooperatives Arbeiten* mittels Computertechnologie unterstützen. Dieser Thematik widmet sich der Bereich der *Computer Supported Cooperative Work* (CSCW). Durch die Einbeziehung von mobilen Endgeräten in die Nutzungsszenarien ist der Bereich der Mobilen Informationsverarbeitung (*Mobile Computing*) betroffen.

Der Lösungsansatz adressiert die Verwendung von Softwareagenten für mobile Anwendungen. Deshalb ergeben sich Fragestellungen aus dem Gebiet der Softwareagenten [WJ95a][WJ95b].

1.4 Zielsetzung

Gegenstand dieser Arbeit ist u.A. die Konzeption einer *Systemarchitektur.* Dazu wird der Begriff der Systemarchitektur definiert:

Definition 2: Die Architektur eines Systems wird beschrieben durch

- *Verfahren* und *Methoden*, mit deren Hilfe die Funktionen der Komponenten realisiert sind sowie
- das *Zusammenwirken* der *Komponenten* im Gesamtsystem und deren Abbildung auf Module.

Das mit der Architektur beschriebene System wird im Folgenden als User-Interface-Agent (UIA) bezeichnet, und ist entsprechend Abbildung 4 in seine Umgebung eingebettet.

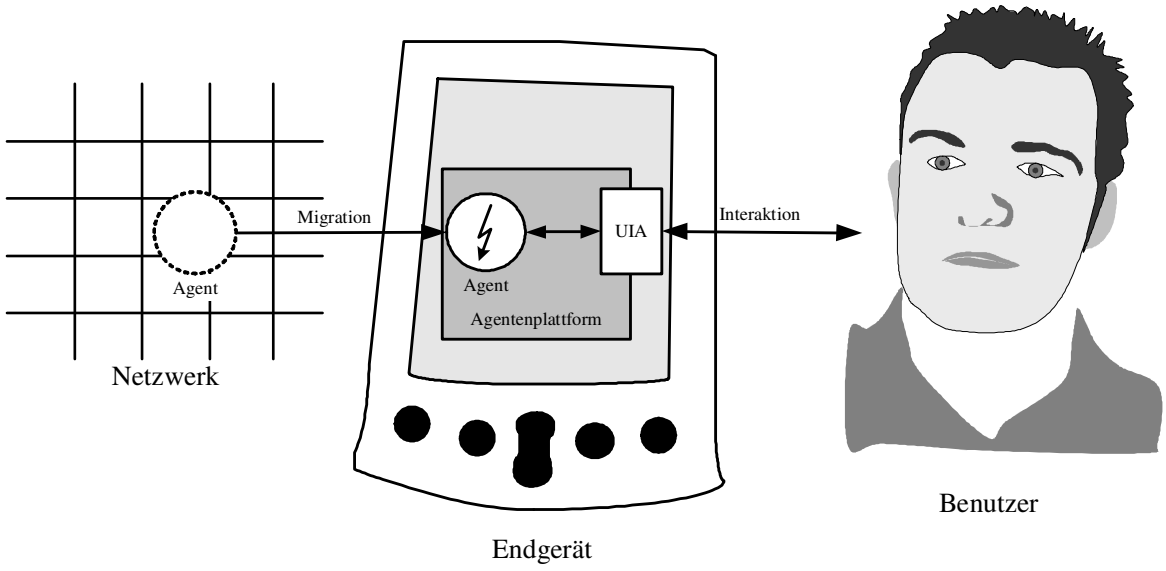

Abbildung 4: Einordnung der UI-Architektur in ein Gesamtsystem

Die zu entwerfende Systemarchitektur soll abstrakt kodierte Benutzungsoberflächen verarbeiten können. Die Abstraktion soll eine *Geräteunabhängigkeit* der Interaktionsdaten bewirken, so dass Benutzungsoberflächen einmal kodiert und vielfach unterschiedlich dargestellt werden können.

Durch Modularität und ein generisches Konzept soll

- der starken Unterschiedlichkeit der Interaktionsschnittstellen und
- dem dynamischen Markt der Endgeräte mit ihren kurzen Produktzyklen

begegnet werden.

Den besonderen Anforderungen an mobile Applikationen (vgl. [BFLM+92] in Abschnitt 1.2.2) und an neue assistenzunterstützende Bedienmetaphern sollen durch folgende Funktionen Rechnung getragen werden:

- *Transfer laufender Dialoge* von einem Gerät auf ein anderes.
- Unterstützung *assistenzgebender Interaktionsverfahren.*
- Verwendung einer *plattformunabhängigen* Basis (Middleware).
- Möglichkeit zum *gemischten Einsatz von mobilen und stationären Endgeräten.*

Die Machbarkeit des Konzepts soll anhand einer prototypischen Realisierung nachgewiesen werden. Die erzielten Ergebnisse sollen dargestellt und kritisch hinterfragt werden, um

festzustellen, welche Verbesserungen noch möglich sind, aber auch um zu ermitteln, welche ansatzbedingten Einschränkungen hinzunehmen sind.

1.5 Vorgehensweise

Der Aufbau dieser Arbeit ist in Abbildung 5 skizziert. Kapitel 2 – *Anforderungsanalyse* – stellt ein repräsentatives Anwendungsszenario für mobile Anwendungen vor. Die Analyse liefert funktionale Anforderungen für die Komponenten eines Lösungskonzepts. Die Kriterien sind am Ende des Kapitels in einem Katalog zusammengefasst, der auch im Ergebnisteil zur Bewertung genutzt wird. Kapitel 3 – *Terminologie und Basistechniken* – führt in die Thematik der Interaktion ein und stellt heute bekannte Modelle für die Mensch-Maschine Interaktion vor. Das Kapitel dient ebenso der Entwicklung einer einheitlichen Terminologie. Heute existente Ansätze, Methoden und Systeme, die dem Anspruch dieser Arbeit teilweise genügen, werden in Kapitel 4 – *Stand der Technik* – vorgestellt. Aus den Modellen für die Mensch-Maschine Interaktion und dem Stand der Technik wird in Kapitel 5 – *Lösungskonzept* – ein Konzept zur Realisierung des User Interface Agenten abgeleitet. Die Umsetzbarkeit des Konzepts in ein konkretes System wird in Kapitel 6 – *Realisierung* – nachgewiesen. Das realisierte System wird in Kapitel 7 – *Vergleich mit existierenden Systemen* – anhand der Kriterien aus Kapitel 2 mit dem Stand der Technik verglichen und bewertet.

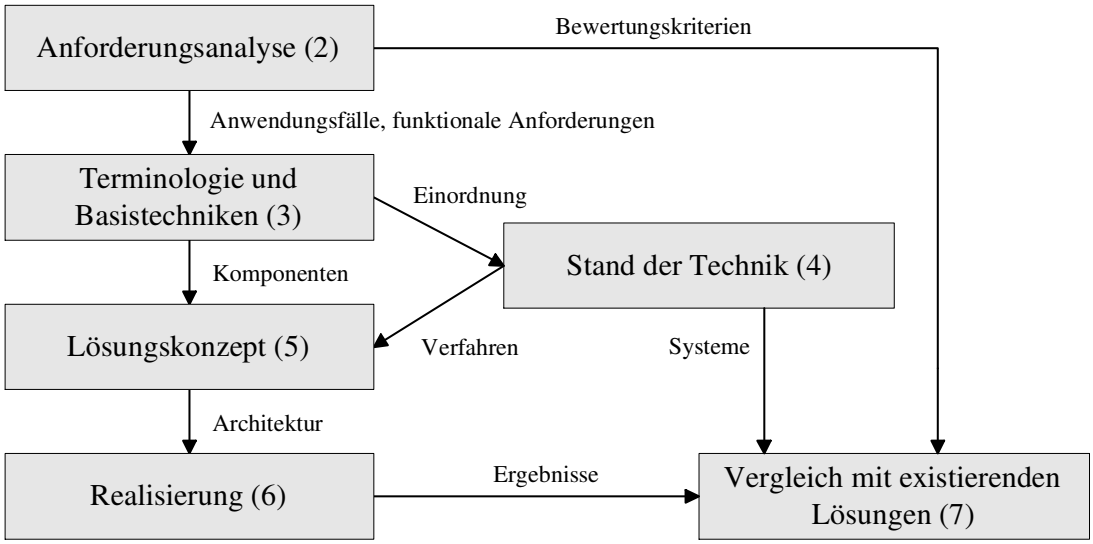

Abbildung 5: Der Aufbau dieser Arbeit

1.6 Ergebnisse

Im Rahmen der vorliegenden Arbeit wurden ein Dialogmodell und eine Architektur für einen User-Interface-Agenten (UIA) erarbeitet. Die gefundene modellbasierte Lösung unterstützt die Entwicklung von mobilen Applikationen nach [BBGM+01]. Es ist möglich

1. Benutzungsoberflächen geräteunabhängig (abstrakt) zu kodieren,
2. die abstrakten Benutzungsoberflächen an verschiedene Endgeräte zu adaptieren und
3. Dialoge kontinuierlich über verschiedene Endgeräte hinweg auszuführen.

Der UIA kann als Teil der Infrastruktur verstanden werden. Er selbst ist nicht mobil, unterstützt aber die mobilen Agenten bei der Interaktion. Durch die Konzeption als *Interaktionsdienste* wurde die Trennung von Anwendungslogik und Benutzungsoberfläche erreicht. Zur Interaktion muss jedes interaktionsfähige Endgerät einen solchen UIA bereithalten.

Das Dialogmodell ermöglicht es, Benutzungsoberflächen einmal zu kodieren und sie auf unterschiedlichsten Endgeräten darzustellen. Das abstrakte Dialogmodell beschreibt die Interaktion auf der Ebene von Dialogschritten, die zur Erreichung von Dialogzielen erforderlich sind. Eine Benutzungsoberfläche wird somit als eine Menge von Dialogzielen beschrieben, die grundsätzlich in einer Hierarchie angeordnet sind. Um die Beschränkungen einer rein hierarchischen Anordnung zu kompensieren, wurden zusätzliche Verweise in die Dialogschritte eingebettet. Die eingebetteten Verweise werden gezielt für die Unterstützung der Interaktion auf kleinen Endgeräten genutzt. Dazu wurde das *Point-of-View*- und *Range-of-View*-Verfahren entwickelt [BS03], das sich als effektiv erwiesen hat. Das Verfahren bringt die Dialogelemente einer Benutzungsoberfläche unter Berücksichtigung der Dialogstruktur in eine Sequenz.

Die Adaption der Endgeräte erfolgt durch die konsequente Einhaltung des Abstract-Factory-Patterns nach [GHJV98]. Hierbei wird aus den Interaktionsdaten ein sog. Dialog-Objekt-Modell generiert. Das Dialog-Objekt-Modell ist eine abstrakte Repräsentation aller Dialogschritte einer Benutzungsoberfläche auf Objektebene. Die abstrakten Dialogobjekte sind mit konkret darstellbaren Dialogobjekten verbunden. Die konkreten Dialogobjekte werden in sog. Rendermodulen erzeugt, die als Factory des o.g. Patterns arbeiten. Die Schnittstelle zwischen abstrakten und konkreten Dialogobjekten dient der Synchronisation der Dialogobjekte sowohl auf der Präsentationsebene als auch auf der Dialogebene (vgl. Abschnitt 3.3.2).

Die Ausführung der Dialoge über verschiedene Endgeräte hinweg ist durch die Speicherung des Zustands der Interaktion auf der Ebene der abstrakten Dialogobjekte erreicht worden. Bei einer Migration wird der Zustand aller abstrakten Dialogobjekte ermittelt und an den migrierenden Agenten übergeben. Der migrierende Agent übergibt die Zustandsdaten an den UIA des Zielsystems, der den Dialogzustand rekonstruiert und ihn fortführt.

Die Verwendung von Rendermodulen ermöglicht es, die Endgeräte als Komposition aus Modalitäten zu behandeln. Durch die Definition einer geeigneten Klassifikation von Ein- und Ausgabemodalitäten und deren Implementation in Form von Rendermodulen ist der UIA sehr einfach an beliebige Engeräte anpassbar.

2 Anforderungsanalyse

In diesem Kapitel wird zunächst ein Szenario für eine mobile Agentenanwendung geschildert. Das Szenario ist repräsentativ und zeigt alle relevanten Anwendungsfälle eines mobilen Agenten auf. Im Anschluss folgt eine Analyse der Anforderungen an eine Infrastruktur, welche die Benutzungsoberflächen automatisch an mobile und stationäre Endgeräte anpasst. Die Analyse bezieht sich auf verschiedene Perspektiven, um ein möglichst vollständiges Bild zu erhalten. Die identifizierten Anforderungen müssen von der zu entwerfenden Architektur erfüllt werden. Darüber hinaus stellen sie Kriterien zur Bewertung von vergleichbaren Architekturen dar und sind am Ende des Kapitels zusammengefasst.

2.1 Anwendungsszenario

Das für die Analyse genutzte Szenario, ist Teil des MAP-Anwendungsszenarios. Das Projekt MAP ("Multimedia Arbeitsplatz der Zukunft") ist eines der sechs Leitprojekte im Bereich Mensch-Technik-Interaktion in der Wissensgesellschaft und wurde durch das Bundesministerium für Wirtschaft und Arbeit (BMWA) gefördert. Das komplette Szenario kann als Animation auf den offiziellen Internetseiten des Projekts angesehen werden [MAP21].

2.1.1 Vom Frühstück zum Büroarbeitsplatz

Das Szenario beginnt am Frühstückstisch. Eine Anwenderin informiert sich über die Aufgaben ihres Arbeitstags und möchte ihr elektronisches Postfach ansehen. Hierfür benutzt sie ihren Personal Data Assistant (PDA). Sie verbindet den PDA über ein integriertes WLAN-Modul mit dem Internet. Der MAP-Agentenserver auf dem PDA stellt eine Verbindung zum MAP-Agentensystem her (Abbildung 6a). Nachdem sich die Benutzerin beim Agentensystem identifiziert hat, wird sie begrüßt. Die Benutzerin beginnt ihre Termine anzusehen und ihre Nachrichten zu lesen. Sie beschließt für die Anreise zu einem Arbeitstreffen in Berlin einen Flug durch einen Buchungsagenten zu reservieren. Dazu gibt sie die Buchung bei dem Buchungsagenten in Auftrag. Nachdem der Buchungsagent verschiedene Angebote recherchiert hat, kann er die Angebote vergleichen und selbstständig eine Reise beim günstigsten Reisebüro buchen.

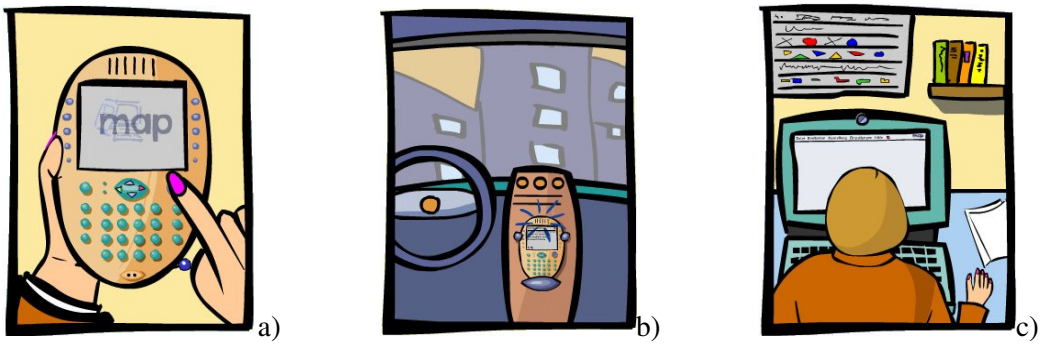

Abbildung 6: Bilder aus dem offiziellen MAP-Projektszenario [MAP21]

Gegen Ende des Frühstücks unterbricht die Benutzerin den Dialog mit dem System, da sie mit dem Auto zur Arbeit fahren möchte. Im Wagen angekommen, verbindet sie den PDA über eine Dockingstation mit der Bordelektronik (Abbildung 6b). Durch die Verbindung erhält das Gerät Zugriff auf ein Mikrofon und einen Lautsprecher, um den Austausch sprachlicher Inhalte und Kommandos zwischen PDA und Benutzerin zu ermöglichen. Die Benutzerin kann sich durch die sprachliche Interaktion auf die Kontrolle des Fahrzeugs konzentrieren und ist nicht durch manuelle Eingaben oder durch das Ablesen ausgaben abgelenkt. Der am Frühstückstisch begonnene Dialog über die Termine des Benutzers wird nun sprachlich fortgeführt. Während des Dialogs informiert das MAP-System die Benutzerin über einen Stau und schlägt eine Ausweichmöglichkeit vor. Die Benutzerin folgt dem Vorschlag und lässt sich durch das Agentensystem entlang der Ausweichstrecke zum Büro leiten. Am Arbeitsplatz angekommen (Abbildung 6c) arbeitet die Benutzerin mit einem Desktop-PC weiter. Nun bestätigt das MAP-Agentensystem die erfolgreich ausgeführte Buchung. Die Benutzerin nimmt diese Information zur Kenntnis und beginnt ihre Arbeit im Büro.

2.1.2 Weitere Anwendungsmöglichkeiten

Das geschilderte Szenario zeigt die Anwendung eines Agentensystems zur Terminverwaltung und Reisebuchung. Darüber hinaus sind u.A. die folgenden Anwendungen denkbar:

- Kommunikation: E-Mail und Online-Chat an beliebigen Orten.
- Brokerage: Aktienkurse abfragen, Aktien kaufen und verkaufen.
- Mobile Banking: Kontostand abfragen und Überweisungen durchführen.
- Informationsdienste: Wetter, Verkehrslage, Kino.
- Einkauf und Bestellung: Bücher, CDs, Tickets.
- Telematikdienste allgemein: Routenplaner, Stadtpläne, Reiseführer.
- Haus-Automatisierung: Fernabfrage und –steuerung von Geräten.

2.2 Anforderungen

Die Anforderungen an eine Systemarchitektur für die Interaktion mit mobilen Agenten lassen sich in vier Klassen einteilen:

- Anwendungsorientierte Anforderungen,
- benutzerorientierte Anforderungen,
- entwicklerorientierte Anforderungen und
- allgemeine Anforderungen.

Anwendungsorientierte Anforderungen folgen aus der Analyse des Szenarios. Aus ihnen können funktionale Komponenten ermittelt werden, die ein mobiles Agentensystem bereitstellen muss, um mit dem Benutzer interagieren zu können.

Benutzerorientierte Anforderungen beziehen sich auf die Bedürfnisse der Anwender, wie z.B. die Nachvollziehbarkeit des Systemverhaltens, Bedienbarkeit, Effektivität, etc. Diese Anforderungen können zusammenfassend auch als *ergonomische Anforderungen* bezeichnet werden.

Entwicklerorientierte Anforderungen beziehen sich auf die Sicht der Entwickler von mobilen Anwendungen der Mensch-Maschine Interaktion.

Allgemeine Anforderungen sind durch Rahmenbedingungen des Desktop- und mobile Computing gegeben, wie etwa durch die Technologie mobiler und stationärer Endgeräte, Standards, etc.

2.2.1 Anwendungsorientierte Anforderungen

Folgende anwendungsorientierte Anforderungen ergeben sich aus dem Szenario an ein mobiles Agentensystem.

- *Offline-Arbeit*
  In vielen Situationen ist es erforderlich ohne eine dauerhafte Netzwerkverbindung mit dem Agentensystem arbeiten zu können.
- *Agentenmigration*
  Agenten müssen ihre Ausführung von einem Gerät auf ein anderes verlagern können.
- *Adaption*
  Die Interaktionsdaten müssen an das jeweilige Gerät und den Kontext adaptiert werden.
- *Agentenkontrolle*
  Zum Steuern der Agenten (z.B. erzeugen, beauftragen, versenden, anhalten, beenden) müssen Funktionen angeboten werden.

- *Agentenstatus*
  Die Existenz von Agenten auf dem Endgerät und ihr Zustand müssen für den Benutzer erkennbar sein.
- *Dialogkontrolle*
  Zur Steuerung (Auswahl, Start, Navigation, Abbruch und Beendigung) müssen Funktionen angeboten werden.
- *Dialogmigration*
  Bei der Migration eines Agenten auf ein anderes Endgerät muss auch der Dialog migriert werden. Dazu müssen Funktionen (anhalten, speichern, wiederherstellen, fortsetzen) angeboten werden.
- *Kontext*
  Das System sollte die Durchführbarkeit von Dialogen im jeweiligen Kontext berücksichtigen. Wünschenswert ist die Möglichkeit, Profile anzulegen.
- *Plattformunabhängigkeit*
  Das Agentensystem muss die verschiedenen Betriebssysteme der Zielsysteme unterstützen.
- *Abstraktion der Interaktionsdaten*
  Für die Kodierung der Interaktiondaten wird eine Beschreibungssprache benötigt, die geräteunabhängig ist und sich nicht an einer bestimmten Technik orientiert.

2.2.2 Benutzerorientierte Anforderungen

Diese Anforderungen beziehen sich auf die Benutzer, deren Bedürfnisse Gegenstand der *Software-Ergonomie* sind. Dabei werden Eigenschaften und Anforderungen der Benutzer sowie technische Gestaltungsmöglichkeiten berücksichtigt. Ergonomische Betrachtungen können z.B. in [Shne99] und [Preec94] nachgelesen werden. Für die Gestaltung ergonomischer Software wurde u. A. die DIN EN ISO 9241 ins Leben gerufen, um eine Grundlage für eine gesetzliche Regelung zu erhalten. Teil 10 dieser Vorgaben setzt sich mit „Grundsätzen der Dialoggestaltung“ auseinander.

Aus der Sicht der Software-Ergonomie besteht Interaktion aus:

- dem Benutzer,
- einer zu erfüllenden Aufgabe,
- einem Computer, der bei der Lösung der Aufgabe eingesetzt wird, und
- einem Kontext, in dem die Aufgabe bewältigt werden muss.

Ein Maß für die Güte einer ergonomischen Software ist die sog. Gebrauchstauglichkeit.

Nach [ISO9241] definiert sich die *Gebrauchstauglichkeit* wie folgt:

> *„Die effektive, effiziente und zufrieden stellende Nutzung eines Produktes gemäß den Erfordernissen des Nutzungskontextes."*

Als Maßstab für die Gebrauchstauglichkeit eines Systems gelten die 7 Grundsätze der Software-Ergonomie:

1. Aufgabenangemessenheit,
2. Selbstbeschreibungsfähigkeit,
3. Steuerbarkeit,
4. Erwartungskonformität,
5. Fehlertoleranz,
6. Individualisierbarkeit und
7. Lernförderlichkeit.

Diese Grundsätze sind Gegenstand des Teils 10 – *Grundsätze der Dialoggestaltung* in [ISO9241]. Zur Einhaltung der Grundsätze werden folgende Techniken vorgeschlagen:

- Redundanz,
- Übersichtlichkeit,
- Nachvollziehbarkeit und
- die Möglichkeit zum Rückgängigmachen von Aktionen.

2.2.3 Entwicklerorientierte Anforderungen

In der Konzeption, Spezifikation und Implementation eines zu entwickelnden Systems stellen sich folgende entwicklerorientierten Anforderungen:

- *Integrationsfähigkeit*
  Die Integration der Architektur in bestehende Systeme sollte möglich sein.
- *Einfachheit*
  Die Erstellung von Benutzungsoberflächen sollte leicht erlernbar sein.
- *Angemessener Aufwand*
  Der Aufwand für die Erstellung einer Benutzungsoberfläche ist dann angemessen, wenn er im Verhältnis zu ihrer Komplexität steht.
- *Ausreichender Funktionsumfang*
  Für die Erstellung von Benutzungsoberflächen sollte ein ausreichend hoher Funktionsumfang vorhanden sein.

- *Modularität und Wartbarkeit*
  Ein System sollte nachvollziehbar in Module gegliedert sein.
- *Isolation der Interaktionsdaten*
  Interaktionsdaten sollten von den funktionalen Komponenten isoliert sein.
- *Erweiterbarkeit*
  Es sollte möglich sein, dem System nachträglich neue Module hinzuzufügen.
- *Portabilität*
  Das System sollte auf alle Zielplattformen portiert werden können.

2.2.4 Allgemeine Anforderungen

Die allgemeinen Anforderungen für die mobile Benutzerinteraktion lassen sich aus den bisher aufgestellten Anforderungen und aus der Motivation ableiten.

Das System soll auch mit mobilen Endgeräten arbeiten können, die über sehr einfache Interaktionsschnittstellen verfügen. Um Komplexe Benutzungsoberflächen darstellen zu können müssen diese mit geeigneten Verfahren in kleinere Teile zerlegt werden (sog. Tailoring).

Mobile Datenkommunikation wird bis zur flächendeckenden Bereitstellung einer breitbandigen Kommunikationsinfrastruktur oft nur geringe Datenraten zu kostengünstigen Preisen anbieten. In vielen Fällen können mobile Endgeräte also nur geringe Übertragungsbandbreiten nutzen. Die Interaktion sollte dadurch nicht gestört sein. Die zu übertragenden Daten müssen ggf. minimiert werden.

Das Systemkonzept sollte sich an *Standards* aus dem Bereich der mobilen Kommunikations- und EDV-Geräte orientieren.

Für die Personalisierung und die Anpassung an häufig verwendete Gerätetypen sollten Standardprofile verfügbar sein.

2.3 Kriterienkatalog

Aus dem Anwendungsszenario und seiner Analyse leiten sich Anforderungen für ein mobiles Agentensystem und insbesondere für die Benutzerinteraktion ab. Die identifizierten Anforderungen berücksichtigen die allgemeine Sicht auf die Systemarchitektur und ihre funktionalen Komponenten, die Sicht des Benutzers und die des Anwendungsentwicklers. Aus den funktionalen Anforderungen leiten sich funktionale Komponenten ab, die später für den Entwurf des Architekturmodells genutzt werden.

Die wichtigsten Anforderungen sind in Tabelle 2 zusammengefasst.

Kriterium	**Anforderung**
Mobilität	*Offline-Arbeit*
	Dialogmigration
	Agentenmigration
Geräteunabhängigkeit	*Plattformunabhängigkeit (OS)*
	Adaption der Benutzungsoberfläche
	Abstraktion der Interaktionsdaten
	Agentenkontrolle
	Agentenstatus
Bedienbarkeit	*Berücksichtigung des Kontexts*
	Dialogkontrolle
	Ergonomie
Sicherheit	*Benutzeridentifikation*
	Agentenidentifikation
	Ressourcenzugriff/Ressourcenschutz
Wartung/Erweiterung	*Modularität*
	Isolation der Interaktionsdaten
	Anpassbarkeit an neue Geräte

Tabelle 2: Zusammenfassung der Anforderungen und Kriterien

3 Terminologie und Basistechniken

In der Vergangenheit hat es zahlreiche Bemühungen gegeben, die Entwicklung von Benutzungsoberflächen zu vereinfachen, da diese einen großen Anteil am Entwicklungsaufwand von Softwareprojekten haben. In [MR92] wird nachgewiesen, dass durchschnittlich 48% des Programmcodes eines Softwareprodukts auf die Benutzungsoberfläche entfällt. Der zeitliche Aufwand beträgt indessen 45% für Design, 50% für Implementation und immerhin 37% für die Wartung.

Die hohen Aufwände bei der Implementation von Benutzungsoberflächen bringen wiederum hohe Kosten mit sich. In [Myer93] werden Probleme bei der Implementation von Benutzungsoberflächen identifiziert, die u.A. auf unvollkommene Designstrategien und iterative Ansätze zurückgeführt werden. Zur Verbesserung der bestehenden Ansätze, wurden Forschungsprojekte initiiert, mit dem Ziel diese Kosten zu minimieren. Als Ergebnis dieser Aktivitäten entstanden zahlreiche Modelle, Architekturen und Frameworks, die Gegenstand dieses Kapitels sind [KB96].

Ausgehend von einer etwas allgemeineren Betrachtung der Interaktion werden in diesem Kapitel Modelle für interaktive Anwendungen vorgestellt. Damit wird eine Basis für vergleichende Betrachtungen entwickelt und ein gemeinsames Verständnis für die verwendete Terminologie sichergestellt. Die Modelle beschreiben die Funktionseinheiten und ihr Zusammenwirken in Systemen und ermöglichen eine Einordnung in die HCI. Außerdem dient dieses Kapitel als Basis für die vergleichende Betrachtung des aktuellen Stands der Technik in Kapitel 4.

Neben den Modellen der interaktiven Anwendungen werden bestimmte Aspekte der Softwareagenten behandelt, die als funktionale Komponenten im Zielsystem arbeiten.

3.1 Interaktion

Interaktion ist ein Kommunikationsvorgang, an dem mindestens zwei Partner beteiligt sind, die sowohl als Sender, als auch Empfänger wirken. Der Empfang einer Botschaft hat eine Zustandsänderung beim Empfänger zur Folge. Der Empfänger wird nun seinerseits zu einem Sender und antwortet auf die empfangene Botschaft, was wiederum eine Zustandsänderung beim Empfänger herbeiführt. Die Interaktion beschreibt also einen Prozess zwischen zwei Kommunikationspartnern, die in einer Wechselbeziehung stehen.

Interaktion kann ein produktiver Prozess sein. Menschen, die miteinander interagieren, können eine Idee besprechen und diese interaktiv verfeinern. Durch Mensch-Maschine Interaktion kann ein Softwareprogramm eingegeben und getestet werden.

Die Kommunikations- bzw. Interaktionsabläufe weisen sich wiederholende Muster auf, wie z.B. die Abfolgen *Frage–Antwort* oder *Aufforderung–Bestätigung–Benachrichtigung*. Solche Muster sind im Folgenden als *Protokolle* bezeichnet. Protokolle stellen eine übergeordnete Sicht auf den wechselseitigen Botschaftsaustausch dar.

In Abbildung 7 ist ein einfaches Protokoll für eine Anweisung dargestellt. Der Initiator eines Protokolls fordert seinen Partner zu einer Handlung auf, der den Erhalt der Botschaft und den Beginn der Handlung bestätigt. Abschließend benachrichtigt der Partner den Initiator über die vollbrachte Handlung.

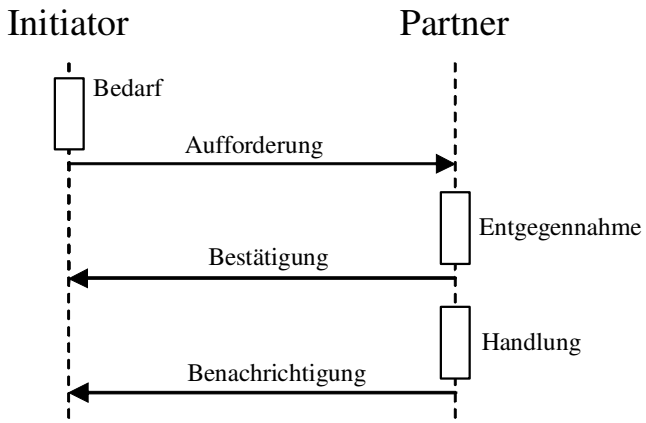

Abbildung 7: Ein Kommunikationsprotokoll

Protokolle eignen sich dazu, gemeinsame Handlungen zu koordinieren bzw. zu synchronisieren. Wenn eine der Botschaften bei der Übertragung verändert wird oder verloren geht, kann dies die Kommunikation empfindlich stören. Kommunikationsprotokolle müssen deshalb an eine Reihe von Randbedingungen geknüpft werden, um die Kommunikation robuster zu gestalten. Solche Randbedingungen sind z.B. eine begrenzte Wartezeit auf eine Antwort oder die Auswertung von Zusatzinformation.

3.1.1 Medium und Modalität

In der Kommunikation ist ein *Medium* der physikalische Träger der Information. Zwischen zwei Kommunikationspartnern muss mindestens ein Medium für den Transport von Informationen vorhanden sein. Die natürliche Kommunikation geschieht vorrangig akustisch und visuell. In der Natur werden eine ganze Reihe weitererer Medien genutzt.

Der Begriff der *Modalität* ist nicht eindeutig einzuordnen. Das Deutsche Wörterbuch [Wahr02] definiert die Modalität folgendermaßen:

Art u. Weise (wie etwas geschieht od. gedacht wird); Ausführungsart....

Bezogen auf die Kommunikation kann die Modalität als Mittel (die Ausführungsart) verstanden werden, welches gewählt wird, um eine Nachricht zu transferieren. Also natürliche Sprache, geschriebener Text, Mimik und Gestik (z.B. ein Händedruck), usw.

Für diese Arbeit definiert Modalität einen konkreten Transportkanal für Informationen, samt spezifischer Sende- und Empfangseinrichtung. Solche sind: Tastatur, grafische Ausgabe auf einem Computermonitor, etc. Ein Medium kann von mehreren Modalitäten gleichzeitig benutzt werden.

Bei der Mensch-Maschine Interaktion werden u.A. folgende Informationsarten verarbeitet:

Art der Daten	**Mögliche Kodierung**
Zeichen oder Text	Telex, ASCII, EBCDIC
Grafik	Videotext : CEPT, NAPLPS, CAPTAIN Bilder: GIF, TIFF, PICT, CGM
Audio	Wave, Rec. ITU-T g.711, MIDI, MPEG/Audio
Fotos	Fax Group 3, JPEG
Videosequenz	Rec. ITU-R 601 and associated audio, MPEG
Video	MPEG, MPEG2
Multimedia	HTML, XML

Tabelle 3: Medien/Modalitäten für die Repräsentation von Daten

Die für die Mensch-Maschine Interaktion gebrauchten Medien können durch den Begriff der Modalität besser unterschieden werden.

3.1.2 Natürliche Kommunikation

Aus der Analyse der zwischenmenschlichen Kommunikation, die im Folgenden auch als *natürliche Kommunikation* bezeichnet wird, können interessante und äußerst wirksame Mechanismen abgeleitet werden, die sich als sinnvoll für die Mensch-Maschine Interaktion empfehlen. Wie verhalten sich aber Menschen, wenn sie kommunizieren?

Bei der natürlichen Kommunikation bestehen einander ergänzende Kanäle, die redundant zusammenwirken und helfen, Inhalt und die Intention des Gegenübers besser und auch unter schlechten Umgebungsbedingungen zu verstehen. Neben akustischen Signalen kommen Mimik und Gestik zum Einsatz.

Bei Wegfall des visuellen Kanals, wie im Falle eines Telefonats, wird sein Einfluss auf das Kommunikationsverhalten deutlich. Ein Partner, der am Telefon länger schweigt, wird nach seiner Hörbereitschaft gefragt, da er nicht gesehen werden kann und intuitiv auf eine fehlerhafte Verbindung oder Abwesenheit geschlossen wird. Solche sinnvollen Reaktionen auf veränderte Randbedingungen werden vom Menschen beinahe intuitiv geleistet, um die entstandenen Lücken – etwa bei Wegfall der visuellen Kanäle – zu schließen. Die noch vorhandenen Redundanzen in der natürlichen Kommunikation reichen aus, um bei leicht geändertem Kommunikationsverhalten ein Gespräch durchzuführen.

3.1.2.1 Phasen

Man kann in einem Gespräch drei Phasen erkennen: Zunächst wird das Gespräch eingeleitet, dann wird über verschiedene Themen gesprochen und schließlich wird das Gespräch beendet. Dieser Zusammenhang ist in Abbildung 8 dargestellt.

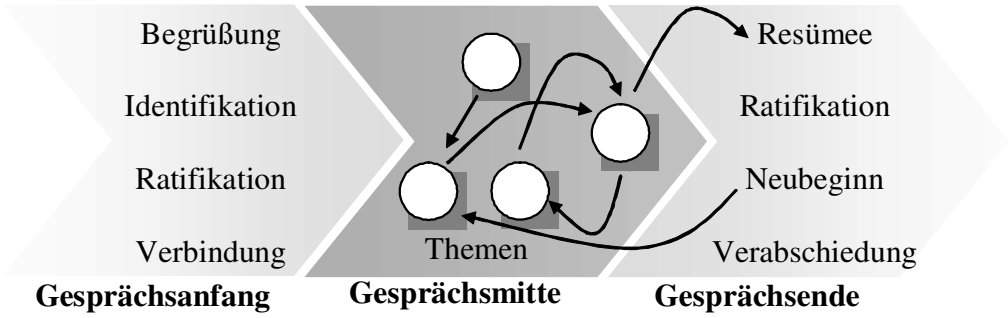

Abbildung 8: Die drei Hauptphasen im Gespräch

Am Beispiel eines Telefonanrufs werden die diese Phasen nun näher betrachtet:

Beim *Gesprächsanfang* meldet sich der Angerufene mit seinem Namen. Die Identifikation ist eine notwendige Gesprächsgrundlage, um sicherzustellen, dass Informationen nur an autorisierte und legitime Personen weitergegeben werden. Die Identifikation wird durch Gegenidentifikation ratifiziert, die folgenden Grüße und Fragen sowie ihre Ratifizierung durch das Gegenüber dienen der Vergewisserung der Gesprächsbereitschaft und der Feststellung der kommunikativen Umstände (wie Zeitdruck, Lärm im Hintergrund etc. als Störquellen).

Schließlich wird der Anrufer auf den Grund seines Anrufs zu sprechen kommen und damit die *Gesprächsmitte* einleiten. Solche Gründe sind z. B. die Reservierung einer Fahrkarte, das Erfragen eines Termins oder eine Verabredung. Mit dem Erhalt der benötigten Information bzw. dem Treffen von Vereinbarungen kann in die Endphase des Dialogs übergegangen werden.

Das *Gesprächsende* hat wie der Gesprächsanfang mehrere Stufen, die nacheinander gewährleisten, dass keine offenen Punkte mehr verblieben sind und die Möglichkeit weiterer persönlicher Kontakte gesichert ist. Abschließend erfolgt das Protokoll der Verabschiedung, das

in mehreren Stufen mit gegenseitigen Wünschen und Schlussworten verläuft. (A: “Bis bald“. B: “OK, bis bald“, A: “Auf Wiedersehen“. B: “Auf Wiedersehen“.)

3.1.2.2 Segmentierung und Verweise

Ein wichtiger Aspekt von natürlichen Dialogen ist der Begriff der *Segmentierung*, die auf unterschiedlichen Ebenen stattfindet. Im Fall geschriebener Sprache bestehen Texte aus Sätzen, Sätze aus Satzzeichen und Worten und Worte aus Buchstaben bzw. Lauten. Analog trennen Redepausen und Stimmhebung die gesprochene Sprache in Segmente; kurze Pausen, Rhythmus, Stimmhebung und Lautwechsel trennen Worte von einander.

Die maschinelle Übersetzung von Texten in verschiedene Sprachen ist abhängig von der Erkennung des Kontexts von Sätzen. Erst die Kenntnis des Kontexts erlaubt die sichere Zuweisung von Subjekt, Prädikat, Objekt, etc. Der Kontext ist meist nicht dem isolierten Satz zu entnehmen, sondern erfordert die Betrachtung der vorangegangenen Sätze. In den Sätzen eines Textes finden sich deshalb Bezüge zum jeweils vorangegangenen Satz. Damit verhält sich ein Text wie eine verkettete Liste von Sätzen, die thematisch und syntaktisch aufeinander aufbauen.

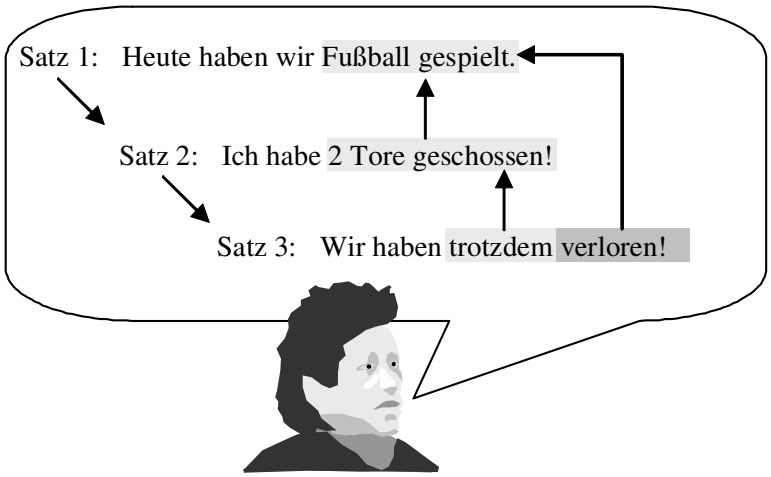

Abbildung 9: Bezüge in Sätzen einer kurzen Erzählung

Das Beispiel in Abbildung 9 erläutert die *Bezüge* anhand einer Erzählung. Der Sprecher beginnt mit dem ersten Satz einen Ausschnitt seines Tages zu erzählen, er hat Fußball gespielt. Der zweite Satz bezieht sich auf den ersten und liefert ein Detail des Fußballspiels, er selbst hat 2 Tore geschossen. Der dritte Satz nimmt wieder Bezug auf den ersten, denn es folgt wieder ein Detail zum Fußballspiel, das Spiel wurde verloren. Der dritte Satz bezieht sich aber auch auf den zweiten Satz, denn das Spiel wurde *trotzdem*, also *trotz der beiden Tore* verloren.

In Abhängigkeit vom Kommunikationskanal bekommt Redundanz eine Funktion: das Gesprochene ist gegenüber dem Geschriebenen flüchtig; im Gesprochenen treten deshalb mehr Redundanzen auf, die ein Textverständnis sicherstellen sollen. Die gesprochene Sprache arbeitet

zusätzlich mit „Schleifenbildung“. Dies ist ein sehr effektiver Prozess, der es erlaubt, jederzeit aus Gesprächsteilen auszusteigen und in andere Gesprächsphasen überzugehen, ohne das Gespräch beenden zu müssen. Aus dem Gesprächsende können die Kommunikationspartner so beispielsweise wieder in die Gesprächsmitte vorstoßen, um ihr Gespräch zu korrigieren, auszuweiten oder um ihre Beziehung neu zu definieren.

Interessant ist zudem die Aktivierung von Verweisen. Ein Zuhörer ist nicht nur ein stummer Empfänger, denn obwohl er keine Worte benutzt, gibt er seinem Gegenüber ständige Rückmeldung über seine Bereitschaft zur weiteren Teilnahme und gibt seine Auffassung über das vom Sprecher Gesagte zu verstehen. Das geschieht mit mimischen und gestischen Mitteln oder durch kurze Laute. Solche Signale werden auch als *Hörersignale* bezeichnet.

3.2 Mensch-Maschine Interaktion

Ein besonders sensibler Bereich des Einsatzes von EDV-Systemen ist immer dann gegeben, wenn Mensch und Maschine in Interaktion miteinander treten. Zu Beginn der EDV-Entwicklung konnten nur ausgewählte Fachleute diese „Dialoge“ führen, denn man benötigte ein erhebliches Fachwissen über die Arbeitsweise der Maschinen. Mit der Massenverbreitung des Computers und dem Einzug in die Arbeitswelt wurde auch der Dialog mit der Maschine einfacher und nachvollziehbarer. Bis heute muss sich aber der Benutzer an die Funktionsweise des programmierten Systems anpassen. Jedoch sollten die Systeme soweit als möglich auf die menschlichen Bedürfnisse eingehen, denn nur so ist die Maschine effizient zu benutzen.

Die Fachdisziplin der Mensch-Maschine Interaktion – *HCI*[8] setzt sich mit der Bedienung von Computersystemen und -programmen auseinander. Ziel ist es den Benutzern die vom jeweiligen System verwirklichten Funktionen einfach und sicher zugänglich zu machen. Die HCI betrachtet nach [HBCC+02] folgende Aspekte:

- (Meta-)Modelle zur Beschreibung der Mensch-Maschine Interaktion
- Einsatz und Kontext von Computern
- Menschliche Eigenschaften
- Architekturen von Computersystemen und ihrer Schnittstellen
- Entwicklungsprozess

HCI ist mit allen Einsatzgebieten von Mikrocomputern konfrontiert: Klassischerweise werden Computer im Büro eingesetzt. Darüber hinaus realisieren sie inzwischen auch Funktionen nahezu beliebiger Geräte. Sie steuern Spielzeuge, Verkehrsampeln und regeln komplizierte chemische und physikalische Prozesse in kleinen und großen Anlagen. Die Mensch-Maschine Interaktion tritt immer dann in Erscheinung, wenn der Mensch auf solche Systeme Einfluss

[8] HCI steht für Human Computer Interaction

nehmen möchte, weil die Ziele des Benutzers kommuniziert werden müssen. Der Kommunikationsprozess stellt gewissermassen eine Hürde dar, weshalb Donald A. Norman [Norm86] spricht auch vom „Gulf of Execution and Evaluation“ spricht. Er verweist auf die „Distanz“ zwischen „Wunsch und Verwirklichung“(siehe Abbildung 10).

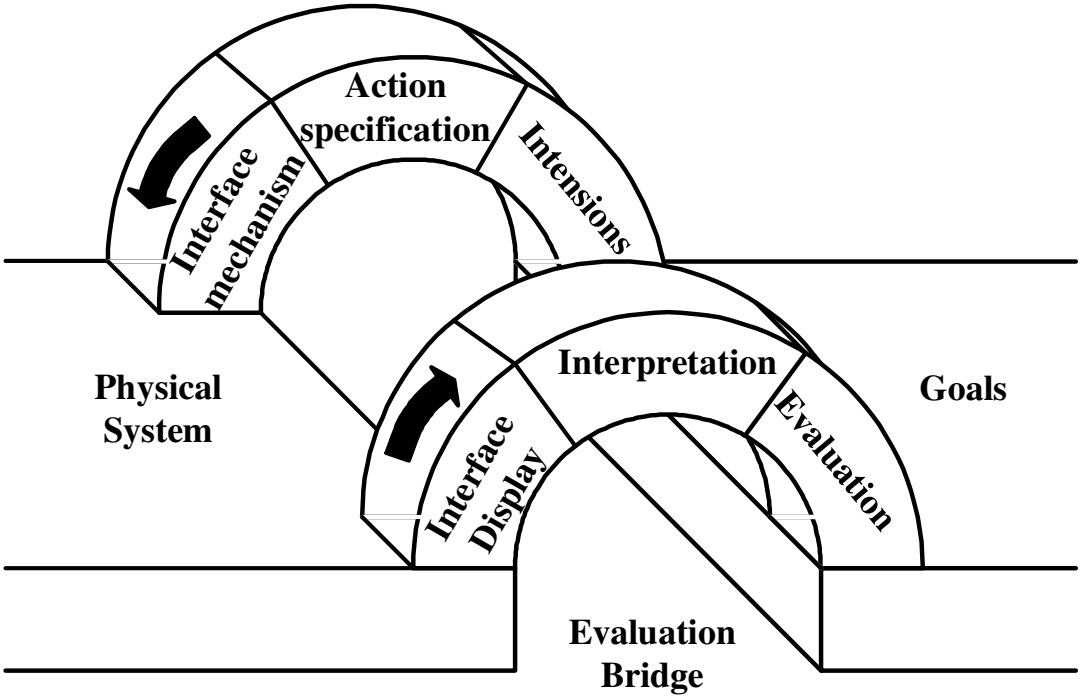

Abbildung 10: Der „Gulf of Execution and Evaluation“ nach [Norm86], S. 40

Analog zu natürlichen Dialogen, kann ein Dialog zwischen Mensch und Maschine als zeitlich begrenzte Kommunikation betrachtet werden, für die eine begrenzte Anzahl von Dialogelementen zur Verfügung steht. Die Dialogelemente entsprechen Handlungsschritten, die im Rahmen der Interaktion auszuführen sind. Dialogelemente enthalten Informationen für den Benutzer über die durchführbaren Aktionen und die Reaktionsmöglichkeiten. Ein Programm bietet die Dialogelemente dem Benutzer an und unterstützt die Eingabe.

Bei der Verhandlung zwischen Mensch und Maschine entsteht somit eine Rollenverteilung, wie sie in Abbildung 11 skizziert ist.

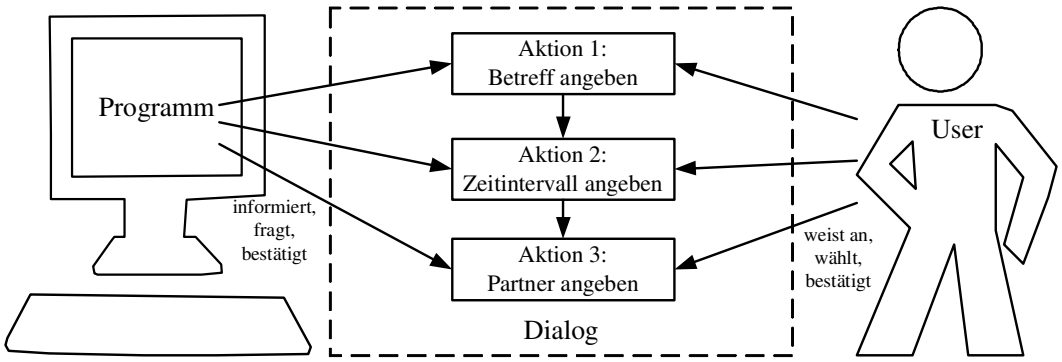

Abbildung 11: Beispieldialog für eine Terminvereinbarung

Die Interaktion zwischen Mensch und Maschine ist auf die Aspekte Dialogkontrolle, inhaltliche Information, Status und Bedieninformation zurückführbar.

- *Inhalt/Zweck:*
  Ist der Teil eines Dialogs, um den sich die Dialogführung dreht. Die Maschine benötigt Informationen vom Benutzer und der Dialog endet im positiven Fall mit der Erbringung der Information an die Maschine durch den Benutzer. Umgekehrt kann der Benutzer im Rahmen eines Dialogs Informationen vom Computer abrufen.
- *Bedieninformation:*
  Soll der Benutzer an einem System Eingaben vornehmen, benötigt er Informationen darüber, in welcher Art die Eingabe erfolgen soll. Bei grafischen Benutzungsoberflächen ist ein Teil dieser Information oft visuell impliziert. Felder zur Texteingabe sind durch ihre visuelle Ausprägung als solche erkennbar.
- *Dialogkontrolle:*
  Der Transfer von Inhalten muss koordiniert verlaufen. Dazu müssen Navigationselemente die Auswahl von Dialogelementen erlauben, Eingaben müssen angefordert, bestätigt oder abgewiesen werden.
- *Status:*
  Bei der Interaktion wird der aktuelle Zustand des Systems dargestellt. Damit wird signalisiert, ob das System auf Eingaben wartet, ob eine Eingabe verarbeitet wurde oder der vorliegende Kommunikationskanal noch aktiv ist.

Die Elemente der Dialogkontrolle sind in Abbildung 12 anhand eines „grafischen Formulars“ zur Terminvereinbarung illustriert.

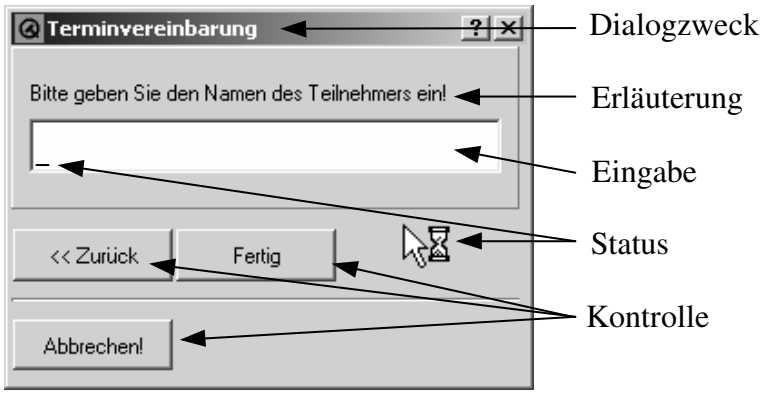

Abbildung 12: Elemente eines graphischen Beispieldialogs

3.3 Modelle für User Interface Systeme

Dieser Abschnitt setzt sich mit verschiedenen Ansätzen zur Modellierung von User Interface Systemen auseinander. Seit der Entstehung insbesondere der grafischen Oberflächen ist deren Entwurf und die Realisierung weitaus komplexer geworden als es zu Beginn der Fall war.

3.3.1 Seeheim Modell

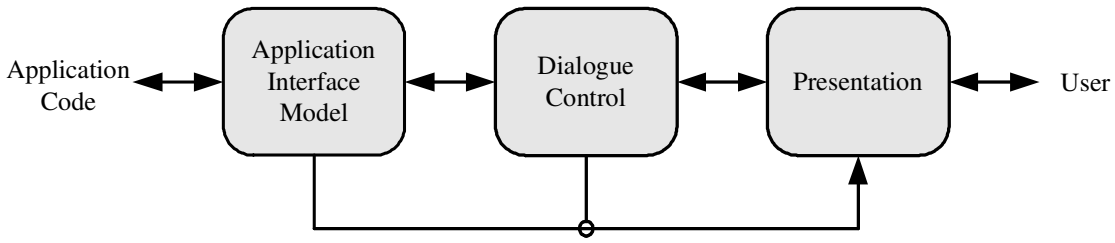

Abbildung 13: Seeheim Modell für UIMS Architekturen nach [Gree85]

Das Seeheim Modell für UIMS-Architekturen in Abbildung 13 wurde 1983 auf einem Workshop in Seeheim entwickelt. Das Modell ist sehr stark vereinfacht und dient nicht der Beschreibung von Struktur bzw. Implementation eines interaktiven Anwendungsprogramms oder seiner Benutzungsoberfläche. Vielmehr werden die Komponenten nach ihren Aufgaben beschrieben. Das Seeheim-Modell setzt sich aus den folgenden drei Komponenten zusammen:

- *Application Interface Model*
  Das Anwendungsprogramm stellt mit seinem Application Program Interface (API) eine Schnittstelle zu seinen Funktionen bereit. Diese Funktionen werden ganz oder teilweise durch das Application Interface Model abgebildet. Das Application Interface Model abstrahiert somit die Funktionalität des Anwendungsprogramms und repräsentiert das Anwendungsprogramm aus Benutzersicht.
- *Dialogue Control*
  Die Dialogsteuerung verwaltet alle Zustände der Benutzungsoberfläche. Alle möglichen Zustände werden definiert und es wird festgelegt, welche Ereignisse zu welchen Folgezuständen führen. Ereignisse können sowohl durch den Benutzer, als auch durch das Anwendungsprogramm ausgelöst werden.
- *Presentation*
  Die Präsentationskomponente ist für die Darstellung der Elemente der Benutzungsoberfläche zuständig. Sie übernimmt ebenfalls die Anpassung der Anwendungsdaten an die Elemente der Benutzungsoberfläche. Auch die benutzerseitigen Ereignisse werden von dieser Komponente an die Bedürfnisse des Anwendungsprogramms angepasst.

Die drei Komponenten entsprechen den semantischen (Application Interface Model), syntaktischen (Dialogue Control) und lexikalischen (Presentation) Aspekten der Interaktion. Das Seeheim Modell ist ein sehr einfaches Modell und geht von einem Wechselspiel zwischen Benutzer und System aus: Der Benutzer gibt ein Kommando, das System führt daraufhin Aktionen aus und antwortet auf das Kommando. Nun kann der Benutzer neue Kommandos geben und weitere Aktionen auslösen.

Der Hauptkritikpunkt am Seeheim Modell ist, dass es nicht detailliert genug ist, um ein UIMS näher zu beschreiben. Zudem haben sich heutige Anwendungsprogramme und ihre Benutzungsschnittstellen zu sehr komplexen Gebilden gewandelt. Dies bedingt den Einsatz systematischer Vorgehensweisen bei der Systementwicklung (etwa Objektorientierung) und macht eine Möglichkeit der Gliederung bzw. der Objektorientierung auch auf der Ebene des UIMS erforderlich. Das Modell berücksichtigt solche Aspekte jedoch nicht, sondern es zentralisiert die syntaktischen Beziehungen einer Benutzungsoberfläche und ignoriert somit die Objektorientierung heutiger Systeme.

Im Bezug auf die Anwendbarkeit des Modells auf ein mobiles UIMS lässt sich anfügen, dass auch die unterschiedlichen Modalitäten der Endgeräte, welche durch eine mobile Anwendung zu bedienen sind, keine Berücksichtigung im Seeheim Modell finden.

3.3.2 ARCH-Modell

Das ARCH-Modell entstand als Ergebnis einer Reihe von Treffen im Rahmen der CHI`1991. Experten auf dem Gebiet der Mensch-Maschine Interaktion suchten ein umfassendes Modell für die Beschreibung von User Interface Management Systemen (UIMS) [BFLM+92]. Bezeichnend

für dieses Modell ist der hohe Abstraktionsgrad der Komponenten von der konkreten Realisierung.

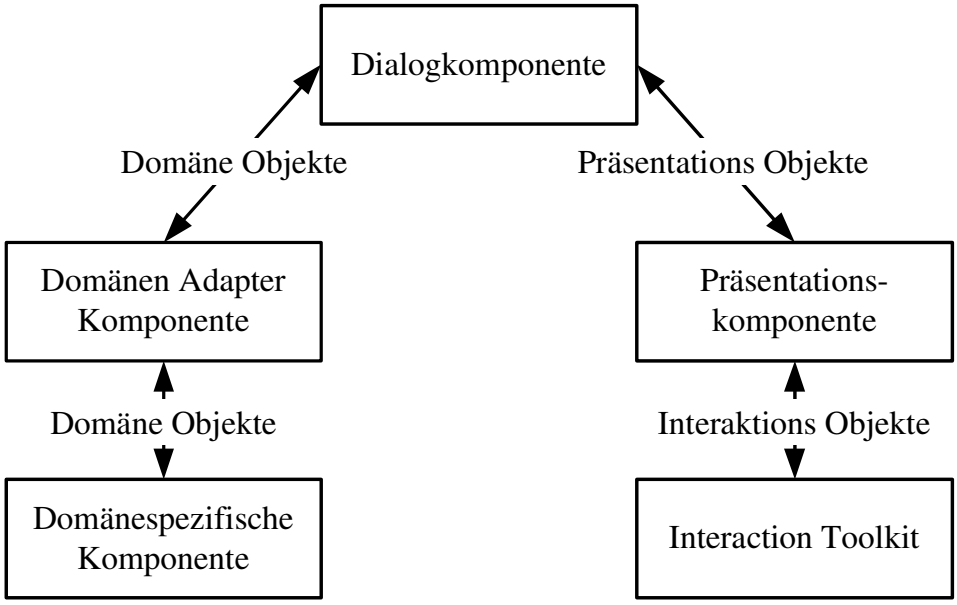

Abbildung 14: Das ARCH-Metamodell nach [BFLM+92]

Das ARCH-Modell ist in Abbildung 14 dargestellt und beschreibt einen Bogen, auf dem alle Komponenten eines interaktiven Systems zu finden sind.

- *Domäne spezifische Komponente*
  Diese Komponente stellt das Anwendungsprogramm mit seinen Funktionen dar, weshalb diese Komponente auch als funktionaler Kern bezeichnet wird. Diese Funktionen sollen durch das UIMS für die Benutzerinteraktion bereitgestellt werden. Sie steht mit der *Dialog Komponente* durch den Austausch von *Domäne Objekten* in Verbindung. Die *Domäne Objekte* werden durch die *Domäne Adapter Komponente* in ein passendes Datenformat gebracht. Die Dialog Komponente und der Kern können unterschiedliche Datenformate verlangen. Wenn keine Anpassung erforderlich ist, weil das Datenformat der beiden Komponenten identisch ist, kann die Adaption entfallen.
- Die *Domäne Objekte*
  repräsentieren die zum Aufruf der Funktionen des Kerns notwendigen Datenstrukturen bzw. die Ergebnisdaten, welche durch den Aufruf generiert wurden.
- Die Aufgabe der *Dialogkomponente*
  ist es, unabhängig von der verwendeten Modalität oder Metapher, die Interaktion zu steuern, Ausgaben zu machen und gewünschte Eingaben anzufordern.

- Die *Präsentationskomponente*
  ordnet den Ein- und Ausgaben passende Darstellungskomponenten für die Präsentation zu.
- Die *Interaction Toolkit Komponente*
  stellt den Teil der Software dar, welcher die Darstellung der Interaktionsobjekte (Buttons, Slider, etc.) vornimmt.

3.3.3 ARCH-Slinky-Modell

Die Gewichtung der Modellkomponenten kann in einzelnen Architekturen sehr unterschiedlich ausfallen, generell können diese aber in das ARCH-Modell eingeordnet werden. Um die unterschiedliche Gewichtung der Komponenten mit in das ARCH-Modell aufzunehmen, wurde es zu einem ARCH-Slinky-Modell expandiert. Metaphorisch wird im ARCH-Slinky-Modell das UIMS als Feder (Abbildung 15) betrachtet, die den ARCH-Bogen aufspannt. Je nach Zustand der Feder können die Bereiche des UIMS unterschiedlichen Raum einnehmen.

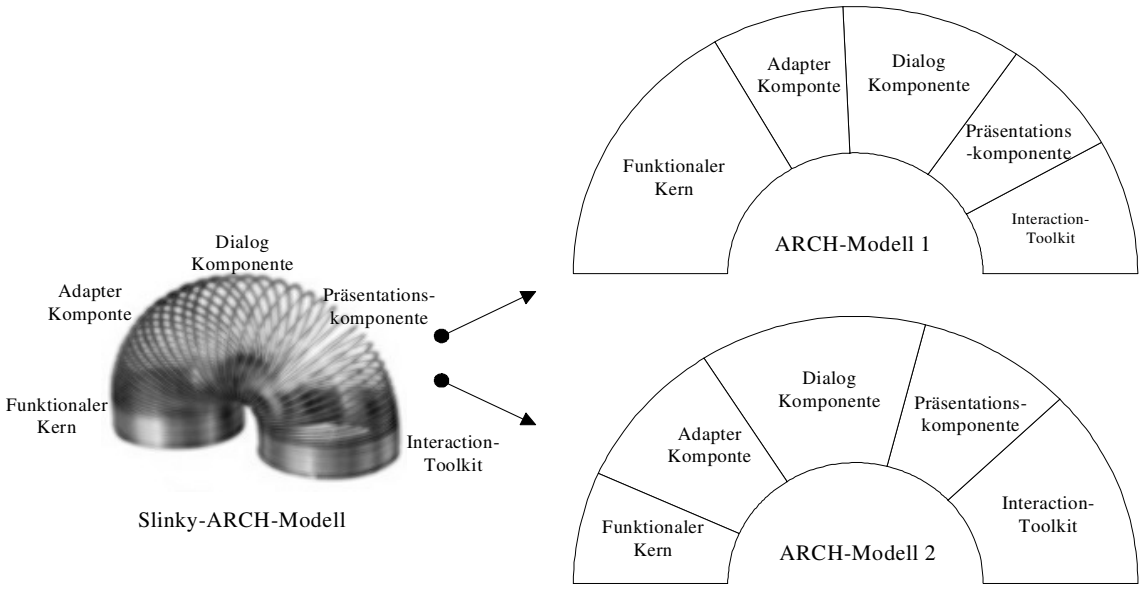

Abbildung 15: Ableitung von ARCH-Modellen aus dem Slinky-ARCH-Modell

3.3.4 PAC-Modell

Ausgehend vom ARCH-Modell dient das PAC-Modell der agentenbasierten Modellierung von UIMS. Es strukturiert ein UIMS in die drei Teile: *Presentation*, *Abstraction* und *Control* (PAC) [Cout87]. Die Teile realisieren die folgenden Funktionen:

- Der *Presentation*-Teil definiert eine konkrete Syntax für die Anwendung, d.h. das Ein- und Ausgabeverhalten der Anwendung, wie sie aus der Benutzersicht beobachtet werden kann.
- Der *Abstraction*-Teil korrespondiert mit der Semantik der Anwendung. Er implementiert die Funktionen, welche die Anwendung ausführen kann. Diese Funktionen sind das Ergebnis einer gründlichen Analyse der durch die Anwendung ausführbaren Aufgaben.
- Der *Control*-Teil sorgt für die Pflege des abstrakten Dialogprozesses und seiner konkreten Präsentation.

Das PAC Modell geht davon aus, dass die Präsentation durch Agenten vorgenommen wird, die mit dem Menschen interagieren. Ein Beispiel für einen solchen interaktiven Agenten ist in Abbildung 16 gezeigt.

PAC-Agenten können durch Komposition zu neuen interaktiven Agenten zusammengefügt werden, wodurch die Erstellung umfangreicher und komplexer Benutzungsoberflächen möglich wird.

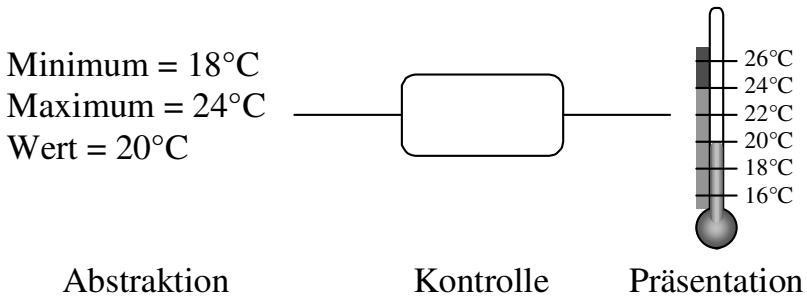

Abbildung 16: Interaktiver PAC-Agent

Grundsätzlich erfolgt die Kommunikation mehrerer Agenten miteinander über die Kontrollkomponente. Abbildung 17 zeigt einen Mehrsichtagenten, der mit zwei Einzelsichtagenten verbunden ist. Der Mehrsichtagent kommuniziert immer dann mit den Einzelsichtagenten, wenn eine Veränderung der Daten durch den Benutzer stattgefunden hat. Daraufhin werden alle mit ihm kommunizierenden Agenten, die dieselben Daten darstellen, veranlasst, die Darstellung zu erneuern.

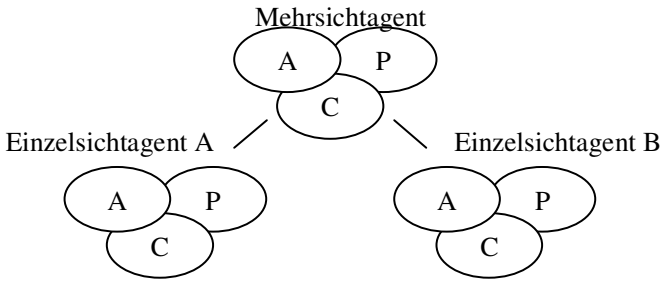

Abbildung 17: Die Verbindung mehrerer PAC-Agenten zu einem Mehrsichtagenten

Das PAC-Modell adressiert die agentenorientierte Systementwicklung unter enger Einbeziehung der Benutzerinteraktion. Es bietet eine konzeptuelle Grundlage zur Modellierung einer Benutzungsschnittstelle, aber keine Hilfe durch schnelles Prototyping oder einer automatischen Generierung der Benutzungsschnittstelle aus der Spezifikation.

Die Vorteile des PAC Modells entsprechen jenen der objektorientierten Systementwicklung, u.A. Wartbarkeit, Modularität und die Konsistenz der entwickelten Systeme.

3.3.5 PAC-Amodeus-Modell

Das PAC-Amodeus Modell kombiniert das ARCH-Slinky Modell mit dem PAC-Modell. Das PAC-Amodeus Modell belegt die Dialog-Komponente des ARCH-Modells mit PAC-Agenten, wie Abbildung 18 zeigt.

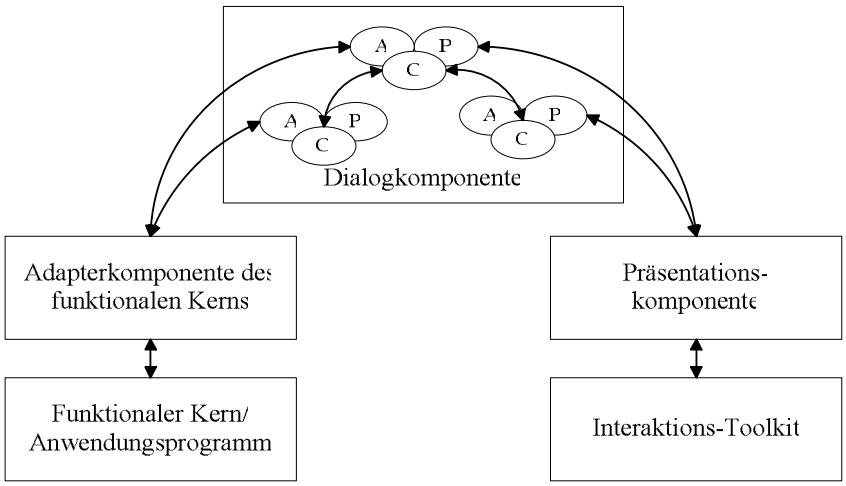

Abbildung 18: Das PAC-Amodeus Modell

Die Dialog Komponente erlaubt zweierlei Kontrollflüsse: Die hierarchische Kommunikation zwischen den PAC-Agenten nach dem PAC Modell und die direkte Verbindung von PAC-Agent zur Adapter Komponente des funktionalen Kerns und zur Präsentationskomponente nach dem ARCH-Modell. Damit ist das PAC-Amodeus ein hybrides Modell, das

- eine Dekomposition nach dem ARCH-Modell erlaubt und
- die Dialogkomponente in einer hierarchischen Weise modelliert.

3.3.6 MVC-Modell

Zur Konstruktion von Benutzungsoberflächen wird heute oft das Model-View-Controller-Modell angewandt, das aus den drei Klassen Model, View und Controller besteht. Vor der Einführung von MVC tendierten Entwürfe dazu, diese Objekte in einem einzigen Objekt zusammenzuführen. MVC modularisiert dagegen den Entwurf von Benutzungsoberflächen und erhöht so die Flexibilität und Wiederverwendbarkeit [KP88][Busc96].

Das objektorientierte MVC-Modell ist dem agentenorientierten PAC Modell sehr ähnlich, da es ebenfalls auf die Dreiteilung der Aufgaben setzt:

- Ein *Model-Objekt*
  stellt die Kernfunktionalität und das Anwendungsobjekt dar (Funktionaler Kern)
- Das *View-Objekt*
  stellt seine Bildschirmrepräsentation dar (Präsentation).
- Das *Controller-Objekt*
  bestimmt die Möglichkeiten, mit denen die Benutzungsschnittstelle auf Benutzereingaben reagieren kann (Dialog).

Ein Anwendungsprogramm, das ein MVC basiertes UIMS benutzt, wird in der Regel viele M, V und C Objekte realisieren. Für eine konkrete Realisierung ergeben sich daher mehrere parallele M-V-C Stränge, die sich über das gesamte ARCH-Modell erstrecken.

Das Ziel der MVC-Architektur ist die Trennung der Verarbeitung von dessen Präsentation und der Steuerung der Benutzerinteraktion. Die Trennung dieser drei Bereiche soll das Gesamtsystem flexibel gegen Änderungen machen. Diese Änderungen können sich auf jeden der drei Teilbereiche beziehen, aber auch auf die Entwicklungsplattform. Abbildung 19 illustriert die Austauschbarkeit anhand des Präsentationsobjekts. Die Zahlenreihe 1,2,3 und 4 kann in Form von Textzeichen, als Balken- oder als Tortendiagramm dargestellt werden.

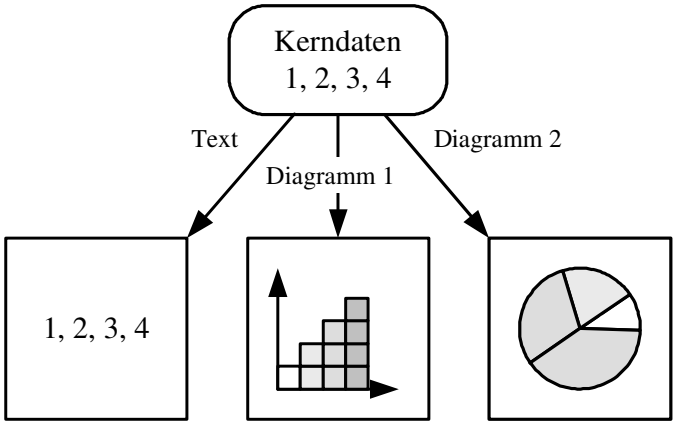

Abbildung 19: Drei verschiedene Präsentationsobjekte für die gleichen Kerndaten

Durch die zunehmende Verwendung objektorientierter Programmiersprachen sind MVC-modellierte Systeme heute oft anzutreffen. Ein Beispiel für MVC ist z.B. das JAVA Toolkit zur Benutzerinteraktion [JAVA].

3.3.7 Pipeline-Modell und CARE-Kriterien

Der Einsatz multimodaler Fähigkeiten in einem Interaktionssystem kann ganz unterschiedliche Ziele verfolgen. Ein wesentliches Ziel ist die qualitative Verbesserung der Interaktion, indem die multimedialen Fähigkeiten moderner PC-Systeme mehr als bisher in die Interaktion integriert werden. Um die Bedienqualität zu erhöhen reicht es nicht aus, mehr Modalitäten zur Verfügung zu stellen, als es bei klassischen UIMS der Fall ist. Eine effektive Verbesserung der Interaktion zu erreichen bedingt die Analyse der Benutzeraktivitäten, um folgerichtige Schlüsse auf die Absicht des Benutzers ziehen zu können. Die durchgeführten Eingaben müssen hierzu in eine Beziehung gesetzt werden.

Ein Beispiel für die Analyse der Benutzereingaben und für die folgerichtige Ableitung einer angemessenen Reaktion ist in [Bolt80] dargestellt. Bolt beschreibt ein Demonstrationssystem, das graphische und sprachliche Eingaben miteinander in Beziehung setzt. In dem bekannt gewordenen Beispiel wird der gesprochene Satz „Put that there!“ in Verbindung mit Mauseingaben gesetzt, die zunächst auf ein Dateisymbol und dann auf einen Ort auf dem Bildschirm verweisen. Das System interpretiert dies folgerichtig als Aufforderung, ein Dateisymbol zu verschieben.

Die CARE-Kriterien [Cout95][NC95] dienen der Sicherstellung der Funktion eines multimodalen UIMS. CARE ist ein Akronym und fasst die Begriffe *Complementary*, *Assignment*, *Redundancy* und *Equivalence* zusammen.

Ausgehend von einer Interaktions-Pipeline, welche die Interaktionsdaten durch eine mehrstufige Kette von Ein- und Ausgabemodulen schickt, können die Daten der Module auf der Eingabeseite

mit den Daten der korrespondierenden Module auf der Ausgabeseite verknüpft werden. Das Pipeline-Modell ist in Abbildung 20 skizziert.

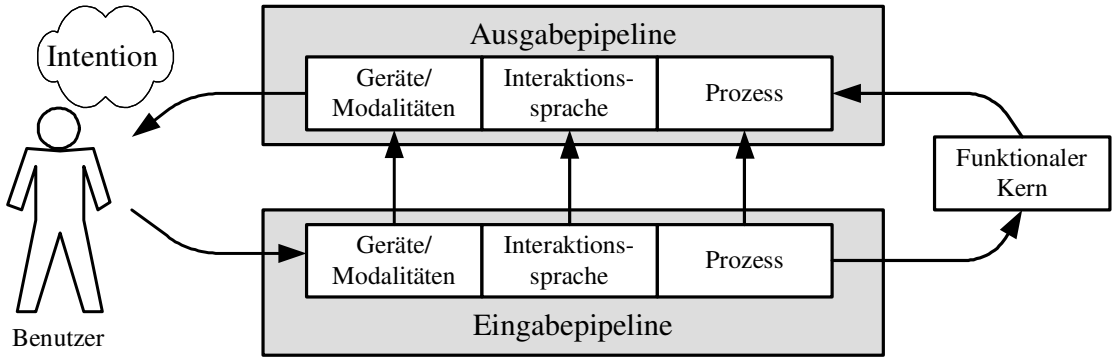

Abbildung 20: Das Pipeline-Modell

Die CARE-Kriterien können auf jede Stufe der Interaktionspipeline angewendet werden, um ein multimodales UIMS bezüglich seiner Anwendbarkeit zu prüfen. Zur Anwendung der Kriterien wird angenommen, dass ein Dialog von einem Zustand A in einen Zustand B überführt werden soll. Dazu stehen sowohl dem Benutzer, als auch dem System verschiedene Ein- Ausgabekanäle zur Verfügung. Die Kriterien liefern eine Aussage darüber, ob und wie der geforderte Zustandsübergang erreicht werden kann.

- Das *Complementary-Kriterium*
  bezieht sich auf die Verfügbarkeit mehrer komplementärer Kanäle, die ergänzend zueinander benutzt werden können.
- Das *Assignment-Kriterium*
  prüft die Existenz mindestens einer Modalität, die den Übergang von A nach B ermöglicht.
- Das *Equivalency-Kriterium*
  garantiert die Verfügbarkeit alternativer Interaktionsweisen zur Erreichung des selben Ergebnisses.
- Das *Redundancy-Kriterium*
  bedeutet eine Verschärfung des *Equivalency*-Kriteriums und bedeutet, dass die Benutzung einer alternativen Eingabemethode innerhalb eines Zeitfensters als Korrektur und nicht als Neueingabe gewertet wird.

3.4 Dialogmigration

Bei der Durchführung von Dialogen auf mobilen Geräten kommt auch die Migration der Dialoge von einem Gerät auf ein anderes in Betracht. Für den Benutzer stellt sich der Anwendungsfall in etwa folgendermaßen dar:

1. Der Benutzer teilt dem System den Wunsch zum Gerätewechsel mit.
2. Zur Migration gibt der Benutzer eine Zieladresse an.
3. Der Dialog wird auf dem bisherigen Gerät geschlossen.
4. Der Dialog wird auf dem anderen Gerät geöffnet und kann fortgeführt werden.

3.4.1 Dialogzustand

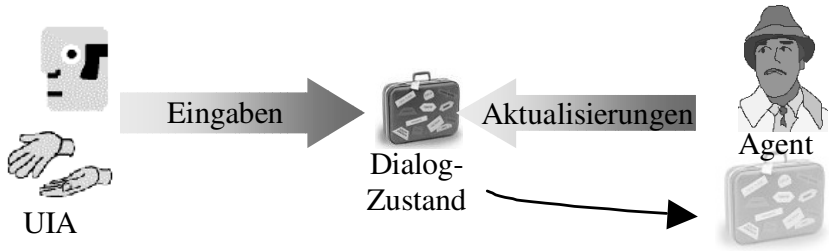

Abbildung 21: Ein Agent nimmt den Dialogzustand mit zum Zielgerät

Zur *Fortführung* des Dialogs wird der Dialogzustand zum Zeitpunkt der Migration ermittelt und auf dem neuen Gerät wiederhergestellt. Der Dialogzustand ist das Ergebnis der Benutzereingaben und der Aktualisierungen, die durch den dazugehörigen Agenten vorgenommen wurden (Abbildung 21).

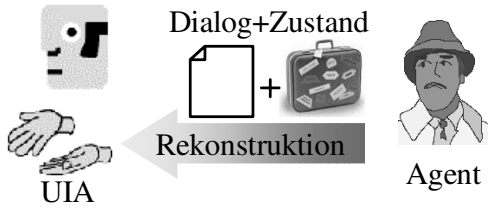

Abbildung 22: Am Reiseziel wird Dialog und Zustand rekonstruiert

Anstelle der Mitnahme des Dialogzustands durch den Agenten könnte der User Interface Agent die Daten selbst zum nächsten UIA für die Dialogausführung übermitteln, um dem reisenden Agenten den Datentransport zu ersparen.

3.4.2 Gerätefähigkeiten

Ein mobiler Dialog kann auf ein Zielgerät treffen, das nicht alle Dialogteile ausführen kann. Die Gerätefähigkeiten müssen deshalb vor Beginn des Dialogs geprüft werden, um eine Fehlerhafte Ausführung zu verhindern oder um die Dialogausführung an die Umstände anpassen zu können. Dazu werden alle Elemente des Dialogs mit den Gerätefähigkeiten verglichen und es kann festgestellt werden, ob die Ausführung eines Dialogs ganz oder teilweise erreicht werden kann. Eine Erreichbarkeit eines Teils der Dialogziele kann von Nutzen sein, wenn Aufgaben zu einem späteren Zeitpunkt fortgesetzt werden sollen.

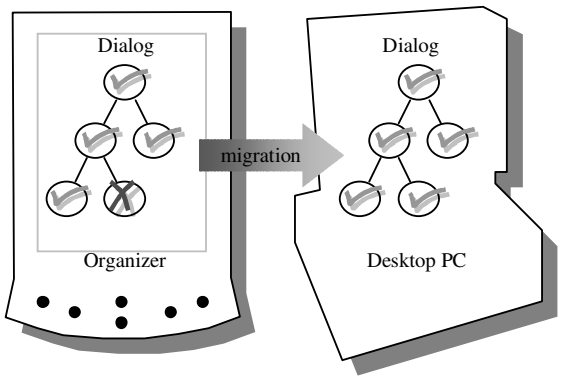

Abbildung 23: Dialogziele können je nach Gerät unterschiedlich erreichbar sein

Insbesondere Recherche- und Suchdienste sollten auf mobilen Endgeräten ausführbar sein, weil hier normalerweise keine umfangreichen Eingaben zu erwarten sind.

3.5 Softwareagenten

Eine einheitliche Definition für Softwareagenten wurde bislang noch nicht vereinbart. Dennoch lassen sich vier wesentliche Merkmale bei den unterschiedlichen Definitionen von Agenten erkennen.

Ein Softwareagent ist

1. *fähig zur Kommunikation*
   Agenten können mit anderen Agenten oder auch mit Menschen kommunizieren.

2. *fähig zur Kooperation*
   Agenten können Aufgaben gemeinsam mit anderen Agenten lösen.
3. *autonom*
   Agenten können ohne direktes Eingreifen des Benutzers agieren und haben ein gewisses Maß an Kontrolle über die eigenen Handlungen und ihren Status.
4. *proaktiv*
   Agenten können nicht nur auf Änderungen in der (Agenten-) Umwelt reagieren, sondern sind darüber hinaus in der Lage, ihre Handlungen zielgerichtet auszuführen.

Zusätzliche Anforderungen an Agenten können z.B. Mobilität und Glaubwürdigkeit sein. Ein mobiler Agent ist demnach in der Lage den Agentenserver zu wechseln, bzw. das Rechnersystem zu wechseln, um z.B. auf anderen Systemen zusätzliche Ressourcen für die Lösung einer Aufgabe zu erhalten. Ein *believable* (glaubwürdiger) Agent verpflichtet sich dazu, Anfragen immer wahrheitsgemäß, d.h. dem Stand des eigenen Wissens entsprechend, zu beantworten und angefragte Dienste immer auszuführen.

Die *Foundation for Intelligent Physical Agents* (FIPA) ist eine Non-Profit Organisation zur Förderung von Standards für die Interoperation heterogener Softwareagenten. Heterogene Softwareagenten sind in diesem Zusammenhang Agenten, die für verschiedene Agentenplattformen entwickelt wurden. Durch die Schaffung von Standards durch die FIPA wird die erfolgreiche Kooperation verschiedener Agenten angestrebt [FIPA].

3.5.1 Agenten im Schichtenmodell

Die Komponenten eines Agentensystems, das für Assistenzfunktionen eingesetzt wird, können in einem Schichtenmodell angeordnet werden. Das Schichtenmodell wird analog zum OSI-Schichtenmodell[9] angewendet und bezieht sich dabei auf die Systemebenen, die in einem Computer vorhanden sind. Diese Ebenen stellen die Hardwareebene, die Betriebssystemebene und die Anwendungsebene dar wie die Einordnung in Abbildung 24 zeigt. Im Schichtenmodell hat jede Ebene Zugriff auf die Funktionen der darunter liegenden Ebene.

Die Hardwareebene umfasst Komponenten wie Netzwerkkarte, Grafikkarte und Eingabegeräte (Maus, Tastatur, etc.) und kann um weitere Geräte ergänzt werden.

Die Betriebssystemebene stellt Systemfunktionen in Form einer API[10] bereit. Die API kann um zusätzliche Dienste erweitert werden, zum Beispiel durch Einbau zusätzlicher Hardwarekomponenten und der Installation entsprechender Treiber, wie etwa durch den Einbau einer Netzwerkkarte.

[9] OSI=Open System Interconnection

[10] API=Application Programme Interface

Der Ebene der Anwendungsprogramme wird die Anwendungssoftware zugeordnet. Hierzu gehören Textbearbeitungsprogramme oder Terminverwaltungssoftware. In diese Schicht kann auch der Agentenserver eingeordnet werden.

Die Agenten sind Teil des Agentenservers und können über die Betriebssystemebene auf benachbarte Anwendungsprogramme zugreifen. Dazu müssen diese Programme eine entsprechende Schnittstelle im Betriebssystem publizieren.

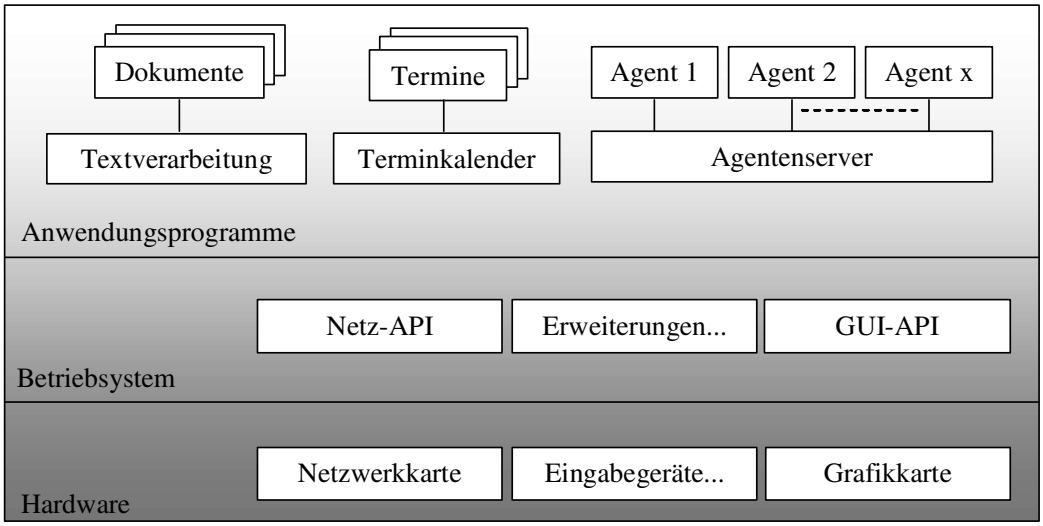

Abbildung 24: Agenten im Schichtenmodell

3.5.2 Mobile Agenten

Mobile Anwendungen bedingen nicht immer die vollständige Übermittlung der Prozessdaten von einem Rechnersystem auf ein anderes. Bei der Prozessmigration sind nach [Vign98] drei unterschiedliche Varianten zu nennen:

- *Remote Evaluation (REV)*
  Ein Programm wird von einem Client auf einen Server übertragen und dort ausgeführt. Das Ergebnis der Ausführung wird zurück auf den Client übertragen. Dies ist etwa bei der Remote Shell auf UNIX der Fall.
- *Code On Demand (COD)*
  Ein Programm wird ganz oder teilweise von einem Server geladen und ausgeführt. Bei Bedarf werden Teile des Programms nachgeladen. Diese Variante ist bei Java Applets realisiert.

- *Mobile Agenten*
  Das komplette Programm wird von einem System auf ein anderes bewegt. Hierbei gibt es keinen bevorzugt Server.

Bei den genannten Varianten der Migration gibt es innerhalb der Mobilen Agenten wiederum unterschiedliche Ausprägungen der Migration:

- Bei der *schwachen Migration*
  werden der Programmcode, die Daten und der Zustand der Daten übertragen.
- Bei der *starken Migration*
  wird zusätzlich der Ausführungszustand zum Zielsystem übertragen.

Mobile Agenten können sich im Netzwerk bewegen, indem sie von einem Agentenserver auf einen anderen wechseln. Die Mobilität muss von der Agentenplattform unterstützt werden und die Agenten müssen die Zustände, die bei der Migration auftreten, berücksichtigen. Vor der Migration muss der Agent seine Daten in einen serialisierbaren Zustand gebracht haben, um sie nach der Migration wieder rekonstruieren zu können. Als *serialisierbar* bezeichnet man Objekte, die in eine Datensequenz überführt werden können, um sie z.B. über das Netzwerk übertragen zu können.

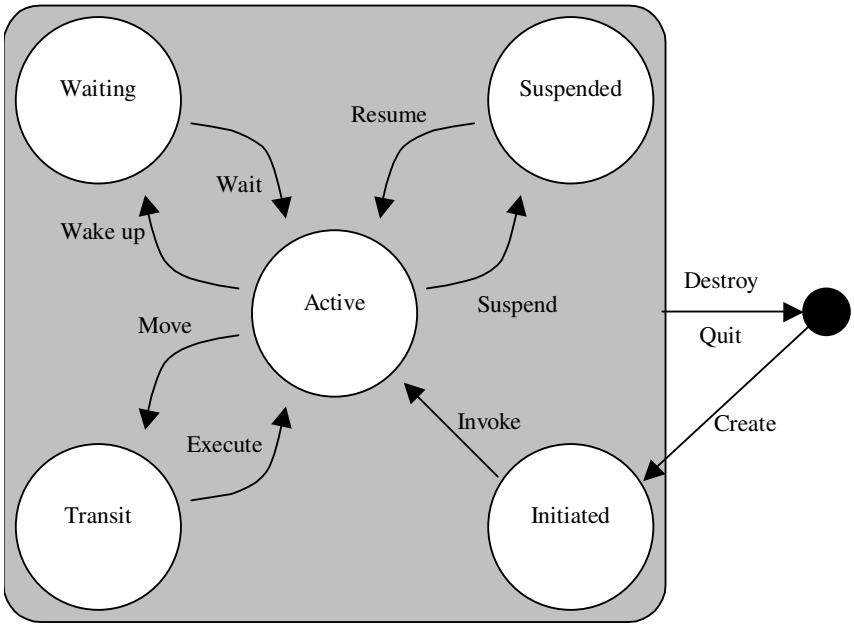

Abbildung 25: Der Lebenszyklus eines Agenten nach [FIPA]

Die oben in Abbildung 25 dargestellten Zustände berücksichtigen die Zustände, die ein mobiler Agent annehmen kann. Den Zuständen kommt die in Tabelle 4 beschriebene Bedeutung zu.

Zustand	Bedeutung
INITIATED	Der Agent wurde soeben erzeugt.
ACTIVE	Agent hat sich bei AMS und DF angemeldet.
SUSPENDED	Die Ausführung des Agenten wurde angehalten.
WAITING	Der Agent ist blockiert bzw. der Prozess schläft, um z.B. auf eine Nachricht zu warten.
DELETED	Agent ist terminiert, nicht mehr beim AMS registriert.

Tabelle 4: Die Zustände des Agent Lifecycle nach [FIPA]

Die Migration eines Agenten kann durch den Agenten selbst oder durch äußere Einflüsse hervorgerufen werden, z.B. wenn der Benutzer die Migration anfordert.

3.5.3 Agentenplattformen

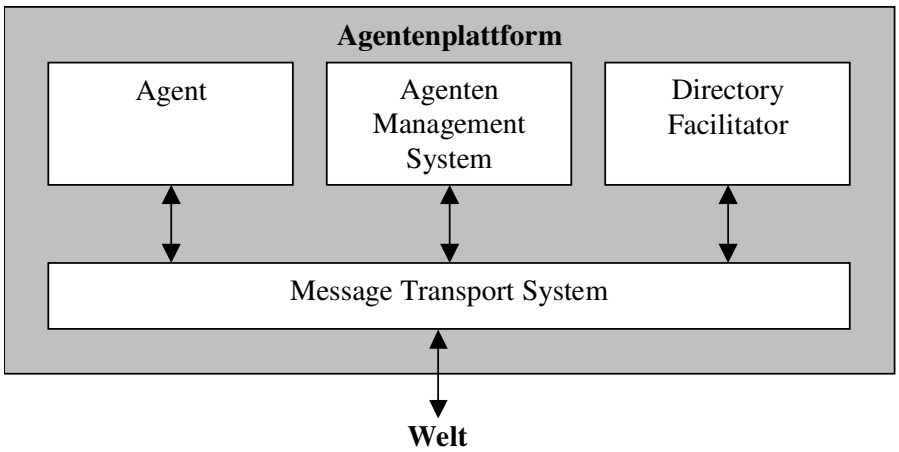

Abbildung 26: FIPA Referenzarchitektur für eine Agentenplattform [FIPA]

Agentenplattformen sollen die Agentenentwicklung unterstützen. Dies geschieht durch

- die Auswahl geeigneter Programmiersprachen,
- das Bereitstellen einer Laufzeitumgebung,
- die Definition von Entwicklungsmethoden und
- das Bereitstellen von Entwicklungs- und Testwerkzeugen.

FIPA beschreibt eine Referenzarchitektur für Agentenplattformen im Sinne einer Menge von Schnittstellen, die eine FIPA-kompatible Agentenplattform implementieren muss.

Die Komponenten der in Abbildung 26 dargestellten Referenzarchitektur erfüllen die folgenden Aufgaben:

- Das *Agent Management System (AMS)*
  ist ein Agent, der die Kontrolle über den Zugriff auf die Agentenplattform hat. Das AMS kann Agenten erzeugen, initialisieren, migrieren und ausführen.
- Der *Directory Facilitator (DF)*
  ist ein Verzeichnisdienst, der die Fähigkeiten und Dienste der Agenten ermittelt und den Agenten darüber Auskunft gibt. Dieser Agent ermöglicht den Agenten die Kooperation durch Funktionen zum Registrieren von Agentendiensten und zum Auffinden von registrierten Diensten.
- Das *Message Transport System (MTS)*
  wickelt den Transport der Agentenbotschaften zwischen den Agenten ab.

Da AMS und DF selbst Agenten sind, basieren ihre Dienste auf dem MTS. Die Kommunikation erfolgt über die logische Wissensrepräsentationssprache *FIPA-SL0*, der *FIPA Agent Management Ontologie* und dem *FIPA Request Interaktionsprotokoll*.

4 Stand der Technik

Um die Basis für die weitere Arbeit zu bestimmen, wird eine Analyse des aktuellen Stands der Technik durchgeführt. Dazu ist das Kapitel in drei Teile gegliedert.

Der erste Teil behandelt Techniken für die Erstellung von Benutzungsschnittstellen. Der zweite Teil beschreibt aktuelle Agentensysteme und prüft ihre Eignung für mobile Anwendungen. Der dritte Teil stellt aktuelle Systeme vor, die das Thema der der Mensch Maschine Interaktion im mobilen Kontext behandeln. Abschließend sind die hier vorgestellten Techniken und Systeme diskutiert.

4.1 Klassifikation

Unter den Techniken zur Erstellung von Benutzungsschnittstellen werden Systeme und Beschreibungssprachen verstanden. Abbildung 27 gibt einen Überblick und zeigt eine Einordnung der Komponenten einer graphisch interaktiven Software in ein Gesamtsystem.

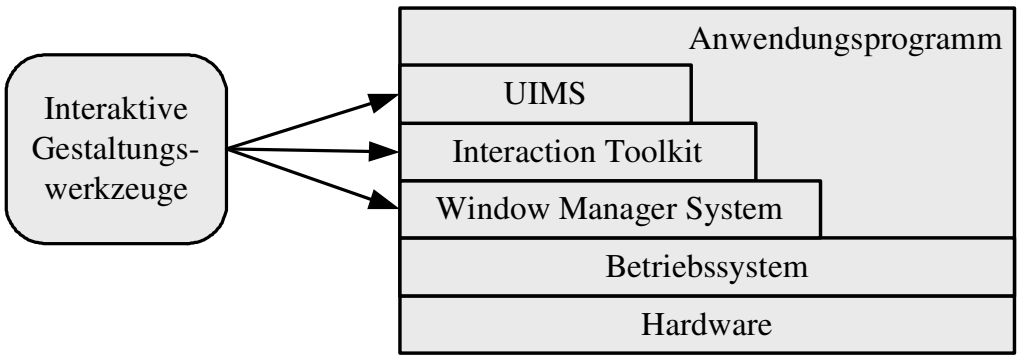

Abbildung 27: Die Schichten einer interaktiven Software nach [FDFH95].

Anwendungsprogramme sowie Interaktive Gestaltungswerkzeuge können Funktionen zur Bereitstellung von Benutzungsoberflächen der Ebenen *Betriebssystem*, *Window Manager System* (WMS), *Interaction Toolkit* (UIT) und *User Interface Management System* (UIMS) zugreifen. Die Interaktiven Gestaltungswerkzeuge erlauben es graphische Benutzungsoberflächen für die Anwendungsprogramme zu erstellen. Interaktive Systeme konzentrieren sich in der Regel auf

eine dieser Ebenen, weshalb das Schichtenmodell geeignet ist bestehende Ansätze zu klassifizieren.

4.1.1 Window Manager Systeme

Window Manager Systeme verwalten den Zugriff auf die Ein- und Ausgabegeräte eines Rechnersystems mit Grafikbildschirm. Sie stellen u.A. so genannte Clipping-Funktionen bereit, welche die grafischen Ausgaben der Anwendungen auf einen zugewiesenen Bereich begrenzen. Da grafische Interaktionsschnittstellen inzwischen standardmäßig Verwendung finden, sind sie meist Teil des Betriebssystems. Als die unterste der drei Schichten ist sie durch einen geringen Abstraktionsgrad gekennzeichnet und daher für diese Arbeit nicht von großem von Interesse.

4.1.2 User Interaction Toolkits (UIT)

User Interaction Toolkits (UIT) abstrahieren Benutzungsoberflächen vom Window Manager System und stellen dem Entwickler höherwertige Interaktive Objekte bereit, die auch als Widgets bezeichnet werden. Mit Widgets ist der Aufbau von Benutzungsoberflächen durch die Komposition sehr einfach möglich. Durch die Verwendung eines UIT reduziert sich der Entwicklungsaufwand einer Softwareanwendung ganz erheblich. Benutzungsoberflächen, die mit dem gleichen UIT erstellt wurden wirken einander ähnlich und verleihen einem Softwaresystem ein konsistentes Erscheinungsbild. Beispiele für Toolkits sind: Das X-Toolkit [Nye90], die Microsoft Foundation Classes (MFC) oder Die Java-AWT Klassen [JAVA].

4.1.3 User Interface Management Systeme

User Interface Management Systeme (UIMS) teilen eine Softwareanwendung sowie deren Entwicklung in Benutzungsschnittstelle und Applikation. Durch die Trennung ist es möglich Modelle als Ausgangspunkt für die Erzeugung von Benutzungsoberflächen zu benutzen (vgl. Abschnitt 4.1.4). Für die Beschreibung von Modell und User-Interface auf Toolkitebene kommen Beschreibungssprachen zum Einsatz. Ein ansatzbedingter Nachteil von UIMS wird besonders bei direkt manipulativen Benutzungsoberflächen wirksam: Da das UIMS zwischen der Applikation und den Interaktionsobjekten der Oberfläche steht, behindert es u.U. die Kommunikation. Vorteile von UIMS sind:

- Austauschbarkeit der Benutzungsschnittstelle für die Anpassung z.B. an die Endgeräte,
- Portabilität der Anwendung, durch die Kapselung vom UIT und
- konsistentes Erscheinungsbild der Benutzungsoberflächen eines UIT.

4.1.4 Modellbasierte UIMS

Bei den modellbasierten UIMS dient ein Modell als Ausgangspunkt für die automatische Generierung von Benutzungsoberflächen (siehe Abbildung 28). Als Basis dienen etwa Domänenmodelle, Benutzermodelle, Kontextmodelle Aufgabenmodelle und Dialogmodelle.

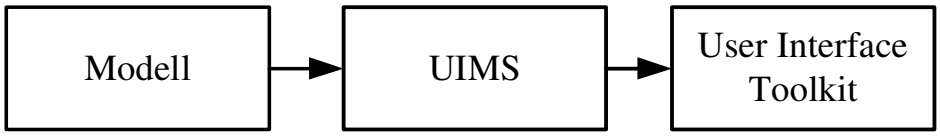

Abbildung 28: Modellbasiertes UIMS

Ein *Domänenmodell* bezieht sich auf begriffliche Grenzen, um die eine Trennung von Funktionalitäten zu gewährleisten. Eine Benutzungsoberfläche kann aus mehreren Teilen bestehen, deren Funktionen aus unterschiedlichen Domänen stammen. Der eindeutig richtige Gebrauch gleichnamiger Funktionen erfordert die Beachtung der Domäne. Unter einer *Domäne* wird ein begrifflich geschlossener Raum verstanden, in dem jeder Begriff eindeutig ist. Die Domäne grenzt ein Begriffsystem gegen andere Begriffsysteme ab, welche die gleichen Begriffe für andere Zusammenhänge gebrauchen können.

Ein *Benutzermodell* stützt sich auf Benutzermerkmale. Merkmale sind etwa Name und Alter die beliebig auf verschiedene Domänen übertragen werden können. Andere Merkmale, wie z. B. spezielles Interessen oder momentane Ziele, können abhängig der Domäne verschieden interpretiert werden. Demnach existieren domänen-unabhängige und domänen-abhängige Teile eines Benutzermodells.

Ein *Kontextmodell* bezieht sich auf den *Kontext* des Benutzers. Es können Kontextinformationen mit Benutzerinteressen verglichen werden, um das Systemverhalten der Situation anzupassen. Wichtige Bestandteile des Kontexts sind Aufenthaltsort und Zeit. Auf diesen Bestandteilen beruhen z.B. *Location Based Services*.

Ein *Aufgabenmodell* wird aufgrund einer der Analyse der vorzunehmenden Aufgaben erstellt und beschreibt die Funktionalität des Gesamtsystems. Formell besteht ein Aufgabenmodell aus den Komponenten: Ziel, Gegenstand, Arbeitsmittel, eine oder mehrere Benutzerrollen und Unteraufgaben mit temporalen Beziehungen. Die Aufgabenmodellierung stellt eine Phase im Softwareentwicklungsprozess vor dem Systementwurf dar.

Ein *Dialogmodell* bezieht sich auf abstrahierte Beschreibungen von Dialogsschritte, die in einer Dialogbeschreibungssprache spezifiziert werden.

4.1.5 Ordnungsschema

Das Modell nach [FDFH95] erlaubt die Ordnung der Techniken nach Schichten.

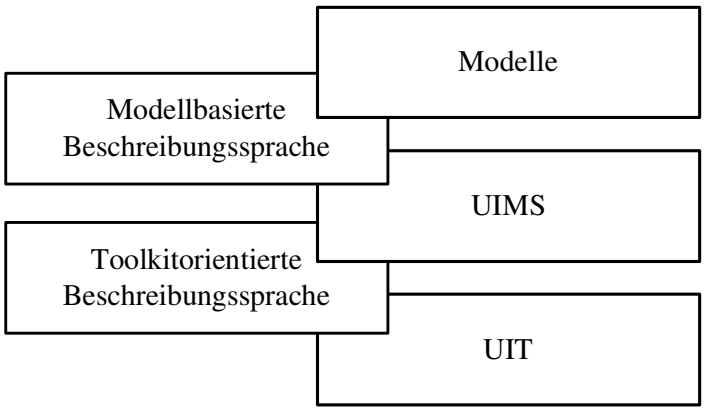

Abbildung 29: Ordnungsschema für Techniken

Theoretisch können alle Bereiche aus dem Schema in Abbildung 29 durch ein System zur Erstellung von Benutzungsoberflächen abgedeckt werden. In der Praxis ist dies jedoch meist nicht der Fall.

4.2 Beschreibungssprachen

In diesem Abschnitt sind aktuelle Beschreibungssprachen für die Erstellung von Interaktiven Anwendungen dargestellt. Eine Einordnung in das Ordnungsschema aus 4.1 und eine Beurteilung der Sprachen findet sich jeweils in einem Abschnitt Klassifikation.

4.2.1 User Interface Markup Language (UIML)

Die XML konforme *User Interface Markup Language* (UIML) wurde mit dem Ziel entwickelt einen offenen Standard für eine XML-basierte Beschreibungssprache für Benutzungsoberflächen zu schaffen, der jedem frei zugänglich ist. Die Motivation besteht darin Werkzeuge für die Erstellung von Benutzungsoberflächen anbieten zu können, die auf verschiedenen Plattformen arbeiten können. UIML ist durch das OASIS-Konsortium [OASIS] standardisiert und hat inzwischen relative Verbreitung gefunden.

Die Arbeiten an UIML begannen im Jahr 1997 mit der Entwicklung einer kanonischen Metasprache, welche die Beschreibung von Benutzungsoberflächen erlaubt, die unabhängig von

der verwendeten Interaktionstechnik und –metapher kodiert werden. Durch diese Abstraktion sollte es möglich sein ein einmal erstelltes Interface auf gänzlich verschiedenen Geräten auszuführen.

Als Metasprache ist UIML nicht an ein bestimmtes Interaction Toolkit gebunden, sondern es kann verschiedene Interaction Toolkits integrieren. Mit UIML können Typen von Interaktionselementen definiert werden, die sich in den adressierten Benutzungsoberflächen wieder finden. UIML enthält außerdem Sprachelemente, welche die Abbildung einer Interaktionsbeschreibung auf existierende Bibliotheken erlaubt [UIML20].

4.2.1.1 Aufbau

Eine UIML-Beschreibung beinhaltet im Wesentlichen zwei Teile:

1. Die abstrakte Beschreibung einer Benutzungsoberfläche.
2. Die Abbildung der Benutzungsoberfläche auf das konkrete Vokabular einer Benutzungsoberflächen-Bibliothek. Dies sind z.B. Java-Swing-Objekte, die angezeigt werden können.

Die abstrakte Beschreibung der Benutzungsoberfläche ist in vier Teile gegliedert.

1. *Die Struktur der Benutzungsoberfläche*
   in der alle ihre Teile klassifiziert und benannt sowie in eine hierarchische Ordnung gebracht sind. Die vergebenen Elementnamen sind dokumentenweit eindeutig und dienen der Referenzierung.
2. *Die Zuweisung von Attributen an die Dialogelemente.*
   Für die Elemente werden zunächst Eigenschaften spezifiziert. Den Eigenschaften werden dann Werte zugewiesen.
3. *Die Definition von Konstanten.*
   Dieses Hilfsmittel kann dazu verwendet werden, schnell und einfach die Sprache der erzeugten Benutzungsoberfläche umzuschalten.
4. *Spezifikation der Verhaltensbeschreibung.*
   Hier wird Bezug genommen auf mögliche Ereignisse und es werden Reaktionen festgelegt. UIML bietet dazu die Definition von Regeln nach dem Schema wenn Bedingung X erfüllt ist, dann führe Aktion Y aus. X kann sich hierbei auf den Zustand von Dialogelementen beziehen. Y kann die Änderung des Attributs eines Dialogelements bewirken, oder die Ausführung einer Methode im Domänenspezifischen Teil des Systems.

Um eine Benutzungsoberfläche in UIML einzukodieren sind strenggenommen alle Teile der Spezifikation erforderlich. Die Verhaltensbeschreibung enthält nur sehr einfache Funktionen und kann ggf. entfallen.

4.2.1.2 Abstraktion

Bei UIML werden die Bedienelemente einer Benutzungsoberfläche durch das jeweils verwendete Interaction Toolkit realisiert. Das Toolkit greift auf eine Programmiersprache, wie Java und seine Bibliotheken zurück. Die Elemente der Bibliotheken sind spezifisch für die Programmierumgebung und berücksichtigen eine bestimmte Interaktionstechnik. Um Abstraktion zu erreichen, kann in einer UIML Beschreibung zunächst mit abstrakten Elementen gearbeitet werden, die auf die konkreten Elemente eines Interaction Toolkits abgebildet werden, wie in Abbildung 30 dargestellt ist.

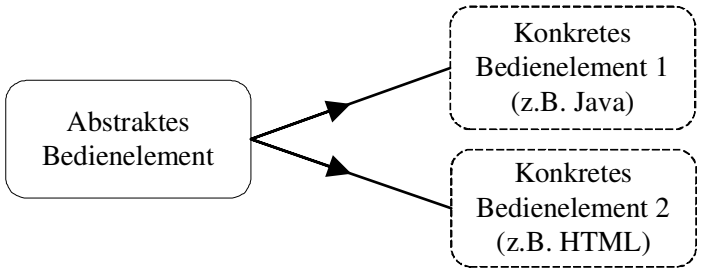

Abbildung 30: Die Abbildung von abstrakten auf konkrete Bedienelemente

4.2.1.3 Klassifikation

Als Metasprache kann UIML eine gewisse Abstraktion von der Präsentation erreichen. Die Regeln für die Abbildung von abstrakter nach konkreter Interfacebeschreibung sind Teil eines UIML Dokuments, womit die Semantik hergestellt ist. UIML gibt keine Regeln für die Dialoggestaltung vor, weshalb dieser Teil vom Entwickler zu leisten ist. Soll ein neuer Interfacetypus durch eine UIML Beschreibung unterstützt werden, muss eine entsprechende Abbildungsvorschrift ergänzt werden. Hierbei ist es nicht immer sicher gestellt, dass die bestehenden Interaktionsdaten für das Zukünftige Gerät geeignet angepasst werden können. Dadurch kann die Unabhängigkeit in bestimmten Fällen verloren gehen.

4.2.2 Alternate Abstract Interface Markup Language (AAIML)

Das V2 technical committee of the InterNational Committee for Information Technology Standards (INCITS) arbeitet an der Entwicklung der Alternate Abstract Interface Markup Language (AAIML). Diese XML konforme Sprache dient zur abstrakten Beschreibung von Benutzungsoberflächen, die auf *Remote Devices* zur Darstellung gebracht werden.

Ein Remote Device ist ein beliebiges Gerät, das zur Mensch-Maschine Interaktion in der Lage ist. Remote Devices werden über eine drahtlose Schnittstelle, wie etwa Bluetooth mit einem *Universal Remote Terminal* (URT) verbunden und damit gesteuert. Die URT dienen in den Anwendungsszenarien der AAIML als Interaktions- bzw. Zugangsgerät zu den Remote Devices.

AAIML kodiert die Benutzungsoberfläche des jeweiligen Remote Devices. Weil die als Terminal in Frage kommenden Geräte sehr unterschiedlich ausgelegt sein können, müssen die Interaktionsdaten weitgehend abstrakt und damit unabhängig von spezieller Hardware gehalten werden [ZVG02].

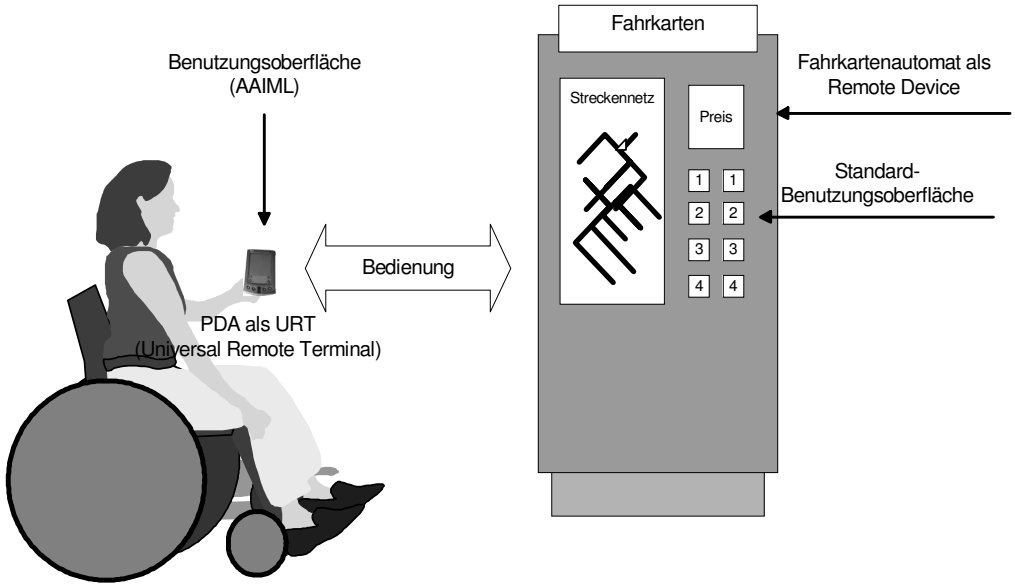

Abbildung 31: Ein PDA als alternative Bedienkonsole für einen Fahrkartenautomaten

In Abbildung 31 ist ein Benutzer gargestellt, der sein eigenes Interaktionsgerät (URT) benutzt, um das Remote Device zu bedienen. Potentielle Profiteure sind also behinderte Menschen, die ihre speziell angepassten Interaktionsgeräte (wie in Abbildung 32) verwenden möchten. Die universelle Bedienbarkeit der Geräte ist gewährleistet, wenn die Interaktionsdaten über die AAIML und ein entsprechendes Austauschprotokoll standardisiert sind.

4.2.2.1 Aufbau

Kern einer URT Spezifikation ist die AAIML zur Beschreibung und Übermittlung von Benutzungsoberflächen. Die Beschreibung wird jedem Remote Device beigefügt. Die Benutzungsoberfläche ist abstrakt definiert, d.h. ohne spezifische Information über bestimmte Modalitäten. Dank der Interfacebeschreibung sind die Remote Devices nun über ein URT ausreichend bedienbar. Die Interfacebeschreibung kann von dem als URT arbeitenden Gerät adaptiert werden. Ein PDA ist z.B. in der Lage grafische Bedienelemente zur Darstellung zu verwenden und per Handschrifterkennung Stifteingaben zu lesen. Für einen Bordcomputer in einem Kraftfahrzeug hingegen, kommen Sprachausgabe und Spracherkennung in Frage, damit der Fahrer nicht von der Beobachtung des fließenden Verkehrs abgelenkt wird. Für blinde Menschen können spezielle Interaktionsgeräte angewendet werden, die haptil zu erfassende Blindenschrift darstellen können wie in Abbildung 32 gezeigt.

Die AAIML definiert einen Satz von abstrakten Interaktionselementen, die als *Abstract Interactor Objects* (AIO) bezeichnet werden und die je für eine bestimmte Ein- oder Ausgabeoperation verantwortlich sind. Auf dem URC wird jedes AIO auf ein konkret darstellbares *Concrete Interactor Objects* (CIO) abgebildet, das in der Lage ist, das gerade verwendete Gerät bezüglich seiner Interaktionsmechanismen zu nutzen. So würde ein Textelement etwa auf dem Display eines PDA durch den CIO visuell dargestellt und auf einem sprachbasierten System akustisch wiedergegeben werden. Das CIO nimmt damit die Adaption der Interaktionsdaten an das jeweilige Endgerät vor.

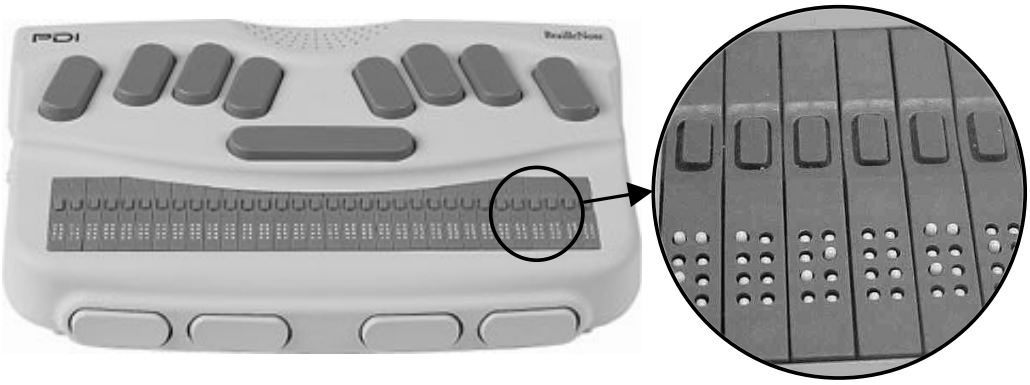

Abbildung 32: Ein Terminal für Blinde Menschen realisiert haptile Interaktion[11]

AAIML basiert auf der Übertragung von Ereignissen, die zwischen dem URT und dem damit Remote Device kommuniziert werden. Die URT Spezifikation baut auf den Mechanismen existierender Netzwerkumgebungen auf, wie z.B. dem Universal Plug and Play (UPnP) von Microsoft und Jini/Java.

Abbildung 33 zeigt die Migration einer Benutzungsoberfläche, wie sie mit AAIML vorgesehen ist. Im Zentrum steht die abstrakte XML-Beschreibung, welche die AIOs einer Benutzungsoberfläche definiert. Die AIOs werden mittels einer XSLT-Datei in CIOs transformiert, die ein bestimmtes Gerät zur Interaktion verarbeiten kann. Die Migration bezieht sich bei AAIML auf die Verlagerung des Dialogs vom Remote Device zum URT hin.

Da die XSLT-Datei das Regelwerk für die Transformation eines AIOs auf einen bestimmten Endgerätetypen enthält, muss eine solche Datei für jeden Endgerätetypus definiert sein, um anpassen zu können.

[11] Es handelt sich hier um das Gerät *BrailleNote BT* (das unter http://www.pulsedata.com zu finden ist)

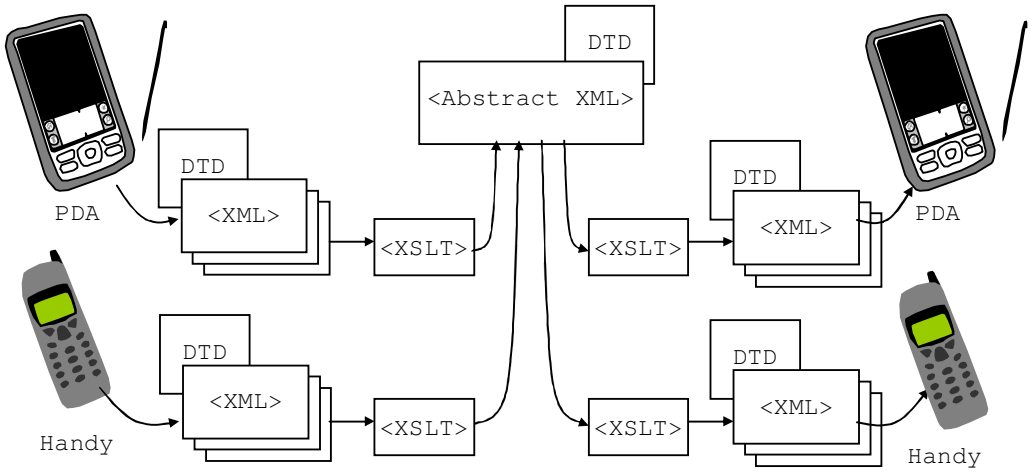

Abbildung 33: Migration von Benutzungsoberflächen mit AAIML nach [LLCR02]

4.2.2.2 Klassifikation

AAIML ist eine Modellbasierte Beschreibungssprache auf Dialogebene. Dem entsprechend können die URT zusammen mit den Remote Devices den UIMS zugeordnet werden.

Die in AAIML abstrakt kodierten Interaktionsdaten auf verschiedenen URT zur Darstellung gebracht werden. Ein URT als Präsentationsgerät für Benutzungsoberflächen muss im Vorfeld nicht bekannt sein. AAIML ist vorwiegend für öffentliche, aber auch privat genutzte Alltagsgeräte vorgesehen, wie etwa Kartenautomaten. Es ist aber auch denkbar die AAIML auf andere Gerätearten anzuwenden.

Ein AAIML-Dokument definiert Objekte, strukturiert sie und weist ihnen Attribute zu. Die Migration einer Benutzungsoberfläche bezieht sich im Falle der AAIML auf die Anpassung einer Benutzungsoberfläche an ein bestimmtes URT und den Transfer der Interaktionsdaten dorthin. Eine kontinuierliche Interaktion über mehrere Geräte hinweg ist aber nicht vorgesehen.

4.2.3 Abstract User Interface Markup Language (AUIML)

Die *Abstract User Interface Markup Language* (AUIML) wurde zunächst bei IBM entwickelt und stellt ein XML Vokabular bereit, das die Absicht des Benutzers bei einem Interaktionsschritt einbeziehen soll. Dies steht im Gegensatz zur konventionellen Gestaltung von Interaktionsschnittstellen, denn konventionelle Systeme fokussieren das Erscheinungsbild einer Benutzungsoberfläche und ihr Verhalten. Der „absichtsbasierte“ Ansatz soll es den Gestaltern bzw. Entwicklern einer Benutzungsoberfläche erlauben sich auf die Bedeutung eines Interaktionsschritts zu konzentrieren, ohne sich um die konkrete Präsentation auf einem bestimmten Gerät kümmern zu müssen. Eine Spezifikation der AUIML findet sich in [AMR00].

4.2.3.1 Aufbau

Grundelement einer AUIML definierten Benutzungsoberfläche ist das *Group*-Tag. Es ermöglicht die Strukturierung der Benutzungsoberfläche indem es seine Teile in eine hierarchische Ordnung bringt.

Die zweite Gruppe von Elementen einer AUIML Beschreibung sind die folgenden Tags, die Datenelemente definieren und deren Typ festlegen:

- Bei einem *Choice*-Element kann der Benutzer eine Auswahl treffen. Die Möglichkeiten der Auswahl kann durch den Parameter *SELECTION-POLICY* innerhalb des Tags angegeben werden. Es ist damit möglich festzulegen, ob ein Element ausgewählt werden kann, oder mehrere, etc.
- <Choice>, <Caption> und <String> um Daten darzustellen.

4.2.3.2 Klassifikation

AUIML ist eine Beschreibungssprache auf der Toolkitebene. Die Sprache verwendet XML-Tags wie <Group>, <Choice>, <Caption> und <String> um Daten zu repräsentieren. Es ist prinzipiell möglich eine Abbildung von UIML nach AUIML vorzunehmen, wodurch es man nicht mehr auf die Verwendung von AUIML angewiesen ist.

4.2.4 XML User Interface Language (XUL)

Die *XML-based User Interface Language* (XUL) wurde im Rahmen des XPToolkit-Projekts für den HTML-Browser Mozilla (Netscape 6) entwickelt [Mozi00]. XPToolkit steht für Cross-Platform Toolkit[12], dem entsprechend wurde XUL dafür entwickelt Benutzungsoberflächen unabhängig von der verwendeten Zielplattform beschreiben zu können. XUL setzt für die Ausführung einer Benutzungsoberfläche den Mozilla Browser als Infrastruktur voraus. Die Präsentation basiert auf CSS1 und CSS2, die Ausführungslogik ist in JavaScript zu implementieren. Beim Start des Browsers wird die XUL-Datei mit der Beschreibung der Benutzungsoberfläche geladen und interpretiert.

4.2.4.1 Aufbau

In jedem XUL Dokument existiert ein *window*-Element, das alle anderen Elemente enthält. Diese Elemente (z.B. *button*) werden mit einem *box*-Element im *window*-Element positioniert. Der folgende Beispielcode erzeugt mit dem Netscape Browser dargestellt ein einfaches Menü mit der Aufschrift „Menuepunkt 1“ und „Menuepunkt 2“ und dazu ein Button-Element mit der Aufschrift „Text des Button 1“. Bei betätigen des Buttons mit dem Mauszeiger erscheint ein Dialog mit dem Hinweis „Sie haben den Button gedrueckt“.

[12] “[..] because X and C look similar if you beat them long and hard with a hammer” [Mozilla00]

```
<?xml version=“1.0“?>
<window
  xmlns=
„http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul“>
  <menubar>
    <menu id=“Menuepunkt1“ label=“Menuepunkt1“ />
    <menu id=“Menuepunkt2“ label=“Menuepunkt2“ />
  <menubar/>
  <box>
    <button id=”button1”
      onclick=“alert(‚Sie haben den Button gedrueckt’);“
      label=”Text des Button 1”
  </box>
</window>
```

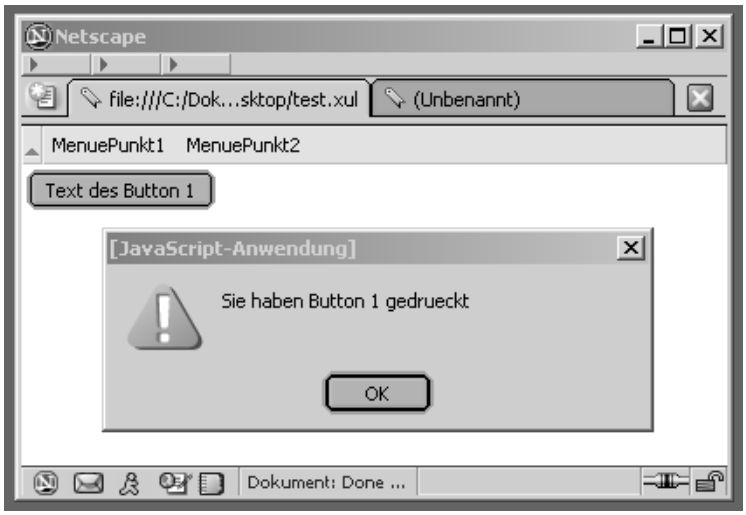

Abbildung 34: Ein Screenshot des XUL-Beispiels

Wie im vorangegangenen Beispiel ersichtlich, ist es möglich außer der Definition des Layouts eine Verhaltensbeschreibung zu integrieren. Wie im dargestellten Fall wird für jede Komponente eine Aktion für ein bestimmtes Ereignis fest. Die Aktion wird durch die Ausführung z.B. eines Java-Skripts festgelegt. Im Beispiel wird bei drücken des Buttons ein Dialog angezeigt.

Dies geschieht im Beispiel durch die Angabe des JavaScript Kommandos „alert([..])“. Die Folge dieses Konstrukts ist allerdings, dass XUL dadurch nicht *sprachunabhängig* ist. Sollen andere Sprachen verwendet werden wie z.B. Visual Basic, muss der Code angepasst werden.

4.2.4.2 Klassifikation

XUL ist eine Beschreibungssprache auf Toolkit-Ebene, der Internetbrowser wirkt hierbei als Präsentationswerkzeug, der Interaktionselemente entsprechend einem Toolkit bereitstellt.

Allgemein können Internetbrowser auch den UIMS zugeordnet werden, wenn man bei der Darstellung von HTML-Seiten z.B. von einem Dokumentenmodell ausgeht. XUL spricht jedoch spezifische Elemente von WIMP-Oberflächen an und ist deshalb als toolkitorientiert anzusehen mit entsprechend niedrigem Abstraktionsgrad.

4.2.5 eXtensible Interface Markup Language (XIML)

Die zu XML konforme eXtensible Interface Markup Language (XIML) zielt darauf ab den gesamten Produktzyklus einer Benutzungsoberfläche zu unterstützen. Die Bestandteile des Produktzyklus sind: Gestaltung, Entwicklung, Management, Strukturierung und Ausführung. Zur Förderung des XIML Standards existiert ein Forum, das sich aus Teilnehmern der Industrie zusammensetzt. Ziele des Forums sind Forschung, Verbreitung, Einführung und Standardisierung der Sprache[Puer02].

XIML soll die Ausführbarkeit von Anwendungsprogrammen auf verschiedenen Software- und Hardwareplattformen ermöglichen. Dazu werden abstrakt beschriebene Interaktionsschritte von zur Laufzeit dargestellten Bedienelementen getrennt.

4.2.5.1 Anforderungen

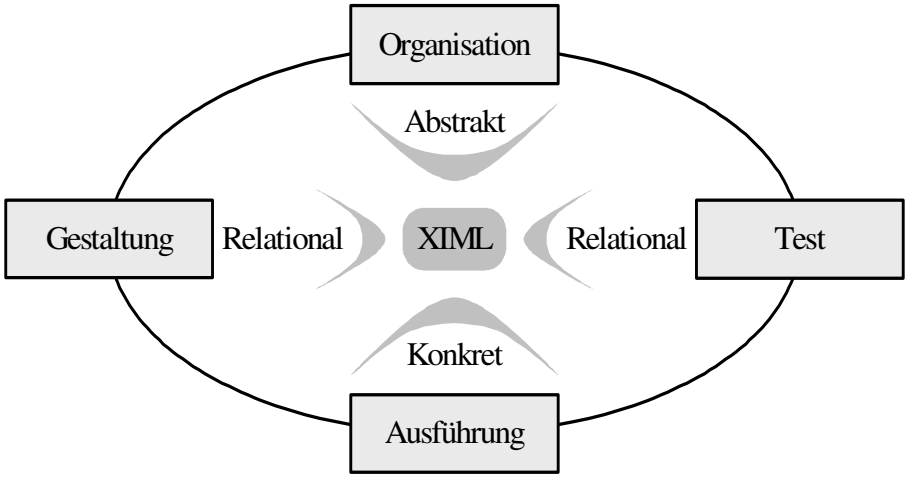

Abbildung 35: XIML berücksichtigt Anforderungen aller Entwicklungsphasen [Puer02]

Der Mechanismus zur Repräsentation der Interaktionsdaten leitet sich aus einer Reihe von Anforderungen ab, die sich auf die Beschreibungsmöglichkeiten der Sprache auswirken, den Anwendungsbereich und die mögliche Unterstützung durch vorhandene Technologien. Das Schaubild in Abbildung 35 fasst die in XIML berücksichtigten Anforderungen zusammen.

Aus den Anwendungsbereichen ergeben sich folgende Anforderungen an XIML:

- *Zentrale Ablage der Daten*
  Die Sprache soll einen verständlichen und strukturierten Mechanismus zur Speicherung der Daten besitzen. Eine zusammengefasste Sammlung an Interaktionsdaten kann etwa bestimmte Anwendungsbereiche abdecken oder allgemeine Interaktionsschnittstellen enthalten. Dies kann in das „Wissensmanagement“ von Firmen eingebunden werden und so die Entwicklung unterstützen.

- *Nachvollziehbarkeit während des gesamten Entwicklungs- und Produktzyklus*
  Die Sprache soll den gesamten Lebenszyklus einer Benutzungsoberfläche unterstützen. Dies schließt Gestaltung, Inbetriebnahme (Operation) und Test (Evaluation) ein. Diese Anforderung ist kritisch, weil ein technischer Rahmen geschaffen werden muss, der die Gebiete Gestaltung, Umsetzung und Test verbindet.

- *Abstrakte und konkrete Elemente*
  XIML soll geeignet sein abstrakte Aspekte einer Benutzungsoberfläche zu beschreiben. Abstrakt wäre hier etwa der Kontext einer bestimmten Aufgabe, in der eine Benutzungsoberfläche verwendet wird und konkret sind z.B. spezifische Darstellungs-elemente die auf dem Bildschirm angezeigt werden können.

- *Beschreibbarkeit von Beziehungen*
  XIML soll dazu fähig sein verschiedene die Elemente, die damit beschrieben werden miteinander in Beziehung zu setzen. Dies ist besonders wichtig um abstrakt beschriebene Elemente auf konkrete Darstellungselemente beziehen zu können.

- *Die zugrunde liegende Technologie*
  Um industriell anwendbar zu sein, soll XIML mit der technischen Umgebung kompatibel sein. XIML darf jedoch nicht von Methoden einer bestimmten Technologie abhängig sein, da sonst die universelle Anwendbarkeit in Frage steht. XIML soll möglichst mit beliebigen Technologien einsetzbar sein.

4.2.5.2 Aufbau

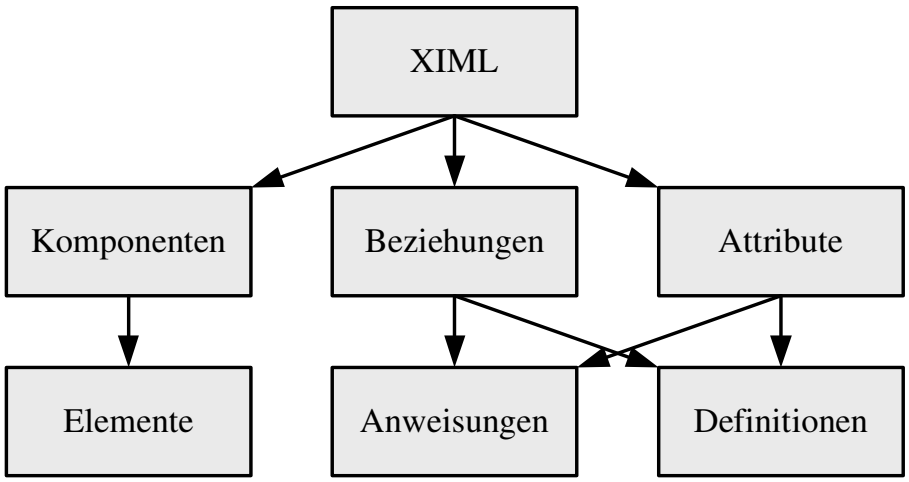

Abbildung 36: Der Aufbau eines XIML-Dokuments nach [Puer02]

XIML stützt sich auf die Beschreibung von Ontologien [Nech91] und auf die Beschreibung von Interface Modellen [Puer97, SPN93, Szek95]. Abbildung 36 zeigt den Aufbau einer XIML Beschreibung.

Im einfachsten Sinn ist XIML eine Ansammlung von Schnittstellenelementen, die aus einer oder mehreren Kategorien von Schnittstellenkomponenten bestehen. XIML ist nicht limitiert bzgl. Der Anzahl der Elemente oder Komponententypen in einer Beschreibung. Praktisch ist aber zu erwarten, dass eine XIML Spezifikation nur relativ wenige Komponenten enthalten wird mit einem hauptsächlichen Elementtyp je Komponente.

In Version 1.0 von XIML sind 5 Komponenten mit der Bezeichnung *task, domain, user, dialog* und *presentation* vordefiniert. Die ersten drei sind als kontextbezogen und abstrakt zu bezeichnen, während die letzten beiden als konkret und der Implementierung zugeordnet werden können.

- *Task*
  Dieses Element enthält den Geschäftsprozess oder die Benutzeraufgabe, welche diese Benutzungsoberfläche unterstützt. Diese Komponente beschreibt eine hierarchische Struktur von Aufgaben und Unteraufgaben und auch den erwarteten *Ablauf* der Aufgaben und dafür benötigte *Attribute*. ...
- *Domain*
  Diese Komponente enthält hierarchisch angeordnete Datenobjekte, die auf einer sehr einfachen Ebene als *Ontologie* [Nech91] begriffen werden können. Elemente werden in diesem Abschnitt eines XIML Dokuments als Objekt-Wert Paare angelegt.

- *User*
  Diese Komponente definiert einen hierarchischen Baum aus Benutzergruppen und Benutzern. Attribut-Wert-Paare legen in diesem Abschnitt eines XIML Dokuments die Gruppen- bzw. Benutzereigenschaften fest.
- *Presentation*
  Diese Komponente definiert eine Hierarchie aus Interaktionselementen welche ihrerseits konkrete Elemente enthalten, die mit dem Benutzer in einer Benutzungsoberfläche kommunizieren. Beispiele hierfür sind Schaltflächen, Fenster, Slider, usw. Im Allgemeinen ist eine relative hohe Granularität erwünscht, so dass die Logik und die Ausführung eines Elements von der Definition getrennt gehalten werden können. Auf diese Weise kann die Darstellung eines Elements ganz dem zugehörigen Rendering System überlassen werden.
- *Dialog*
  Diese Komponente legt eine strukturierte Sammlung von Elementen an, welche Interaktions-Aktionen definiert. So z.B. ein Click mit der Maus, eine gesprochene Antwort oder eine erkannte Geste des Benutzers. Die *dialog* Komponente legt auch den Ablauf zwischen den möglichen Aktionen fest und damit die mögliche *Navigation* in einer Benutzungsoberfläche.

Die vordefinierten Elemente von XIML gehen auf umfangreiche Studien von Arbeiten zur Interface Modellierung zurück [Purta97, SPN93]. Es wurden noch andere Komponenten durch die Forschung als nützlich erkannt, wie etwa eine *workstation* Komponente, die Charakteristiken einer Arbeitsplatzumgebung beschreibt. Diese Aspekte können in den meisten praktischen Situationen durch die existierenden Komponenten in XIML aufgefangen werden können. XIML erlaubt bei Bedarf das Hinzufügen von Komponenten.

4.2.5.3 Relationen (Beziehungen)

Die Datenelemente, welche durch die Komponenten eines XIML Dokuments beschrieben werden, stellen explizites Wissen über eine bestimmte Schnittstelle dar, das Strukturierungs- und Wissensmanagement Funktionen unterstützen kann. Ein wesentlich aufwendigerer Teil einer XIML Beschreibung behandelt die Relationen (Beziehungen), welche zwischen den Elementen einer XIML Beschreibung bestehen. Eine *relation* Komponente verweist auf zwei Elemente in der Beschreibung und legt für die beiden Elemente eine Beziehung fest. Eine solche Beziehung könnte z.B. folgendermaßen aussehen: „Datentyp A *wird angezeigt mit* Datentyp B“. Hierbei werden die Elemente A und B miteinander in eine Beziehung gesetzt.

Die explizite Beschreibung von Beziehungen in einem XIML Dokument stellt eine Wissensbasis bereit, die bei der Gestaltung, Ausführung und Test einer Benutzungsoberfläche verwendet werden kann. Im Einzelnen ermöglicht diese explizite Beschreibung die wissensbasierte Unterstützung von Benutzerinteraktion. Eine detaillierte Darstellung hierzu findet sich in [Puer99].

XIML ermöglicht die Definition von Relationen in einer kanonischen Form und Relations – Anweisungen zur Spezifikation von Instanzen von Relationen. XIML definiert jedoch keine Semantik zu den Relationen. Dies wird den Anwendungen, welche auf XIML zurückgreifen, überlassen.

4.2.5.4 Attribute

Das *attribute* Element erlaubt die Definition von Eigenschaften einer Komponente in XIML. Dazu wird der Name eines Attributs angegeben und es wird ihm ein Wert zugewiesen. Der Wert eines Attributs kann einem grundsätzlichen Datentyp aus XIML entsprechen oder aber die Instanz eines andern Elements sein. Mehrfache Werte sind ebenso möglich wie Aufzählungen und Intervalle. Die Wertzuweisung erfolgt durch die Angabe von Schlüssel-Wert Paaren (Key-Value). Es ist hinzuzufügen, dass Relationen zwischen Elementen auf der Attributebene oder auf der Elementebene hergestellt werden können.

4.2.5.5 Multiplattform Entwicklung von Benutzungsoberflächen

Eine wichtige Anwendung von XIML besteht in der Möglichkeit Benutzungsoberflächen für verschiedene Plattformen und Geräte zu unterstützen. XIML bietet die Möglichkeit mit einer Definition einer Benutzerschnittstelle die Interaktion auf verschiedenen Endgeräten auszuführen. Dies ist möglich durch die Trennung der abstrakten Beschreibung der Interaktionselemente und deren Präsentation auf dem Zielsystem.

Der Aufbau einer Multi-Plattform Beschreibung einer Benutzungsoberfläche für einen Internet-Browser und einen PDA ist in Abbildung 37 dargestellt. Für jedes Endgerät ist eine Präsentationskomponente spezifiziert. Diese Komponenten beschreiben welche Darstellungs-elemente auf einem bestimmten Gerät für einen bestimmten Datentyp zu nutzen sind. Die abstrakte Beschreibung der Benutzungsoberfläche kann so für das jeweilige Endgerät konkretisiert werden.

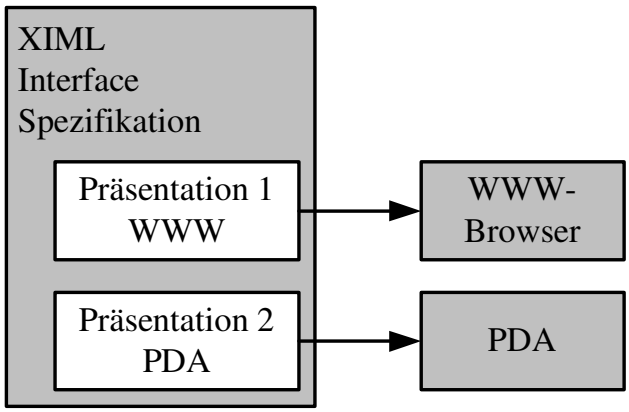

Abbildung 37: XIML beschreibt Benutzungsoberflächen verschiedener Endgeräte [Puer02]

4.2.5.6 Klassifikation

XIML ist eine komplexe modellbasierte Beschreibungssprache, die den gesamten Lebenszyklus einer Applikation berücksichtigt.

Die Modelle unterstützen im Wesentlichen die Bedürfnisse bei der Entwicklung von klassischen Anwendungsprogrammen für mobile oder stationäre Endgeräte. Die Modelle werden jedoch nicht genutzt, um eine automatische Adaption zu erreichen. Die in XIML vorgesehene Methode sieht die abstrakte Kodierung von Interaktionsdaten und die Angabe einer Abbildungsvorschrift für eine konkrete Präsentation vor.

Durch die Notwendigkeit jede Präsentationsform innerhalb eines XIML Dokuments zu konkretisieren, entsteht ein unerwünschter Overhead in den Interaktionsdaten einerseits. Andererseits bedeutet dies, dass im Vorfeld der Entwicklung jedes zu unterstützende Gerät bekannt sein muss. Ohne dieses Vorwissen ist es erforderlich die Interaktionsdaten nachträglich zu ergänzen, wenn ein neuartiges Gerät mit den Interaktionsdaten einer alten XIML-Beschreibung betrieben werden soll.

4.2.6 Wireless Markup Language (WML)

Die *Wireless Markup Language* (WML) basiert auf XML und spielt bei der Nutzung des WAP die zentrale Rolle. WAP steht für *Wireless Application Protocol* und ist eine Technologie, die speziell für mobile Geräte geschaffen wurde [OMA01]. WAP basiert auf einer Client-Gateway-Server-Architektur, entsprechend einem WWW-Server, der Inhalte über das HTTP-Protokoll zugänglich macht. Das WAP-Gateway setzt Anfragen vom mobilen Endgerät auf HTTP um, beschafft den angefragten Inhalt, codiert ihn für das Endgerät und überträgt ihn zum Client.

4.2.6.1 Aufbau

Beim Einsatz von WAP sind die WML, die dazugehörige Skriptsprache *WMLScript* und das Bildformat *WBMP* von besonderem Interesse. Diese Inhaltstypen werden über sog. MIME-Types spezifiziert. Die Serverseite von WAP unterscheidet also wenig von anderen Dokumententypen im Internet. Die Inhalte werden auf dem Server abgelegt oder mittels serverseitigen Technologien, wie z.B. CGI dynamisch generiert.

Die Wireless Markup Language (WML) ist konform zu XML und beschreibt Dokumente. WML bietet die Möglichkeit zur formatierten Ausgabe von Daten, zum Einlesen von Benutzereingaben und zur Navigation in den Inhalten.

Mit WMLScript steht eine Skriptsprache zur Verfügung. Die Sprache ist auf Geräte mit begrenzten Ressourcen optimiert. Der Entwickler schreibt ein Skript, das auf dem Server als Dokument gespeichert wird. Der Compiler befindet sich im Netz (normalerweise im Gateway) und übersetzt das Script in Bytecode, der dann an das Endgerät übertragen wird. Der Zugriff auf spezifische Funktionen des Mobiltelefons, wie z.B. das Initiieren eines Gesprächs, wird über die Wireless Telephone Architecture (WTA) ermöglicht.

4.2.6.2 Klassifikation

WAP ist eine Beschreibungssprache auf der Toolkit-Ebene und darauf spezialisiert Geräten mit sehr kleinen Displays den Zugang zum Internet ermöglichen.

Neben dem Potential des weltweit unbeschränkten Zugangs zu Informationen ergeben sich aber auch Probleme, insbesondere hinsichtlich der Bedienoberfläche. Typische Displaygrößen von Mobiltelefonen liegen bei ca. 100 x 70 Pixel, womit die Menge der Information, die sich darstellen lässt, sehr begrenzt ist. Überlegungen, die Anzeige zu vergrößern, um mehr Information darstellen zu können, sind konträr zum allgemeinen Trend im Design der Miniaturisierung mobiler Endgeräte. Das Angebot von Geräten auf denen WAP-Browser eingesetzt werden, reicht von Handheld Computern mit halber VGA-Auflösung (z.B. Ericsson MC218, Psion, 640x240 Bildpunkte), über PDAs (z.B. Palm *Tungsten C* 320x320 Bildpunkte) bis hin zu Mobiltelefonen und Armbanduhren (z.B. SonyEricsson K50, 160x126 Bildpunkte). Betrachtet man das Verhältnis der Bildschirmfläche von 24:1 so erkennt man, dass eine Entwicklung von Anwendungen, die sinnvoll auf allen Geräten angezeigt werden können, mit besonderen Schwierigkeiten verbunden ist. [SGBF01].

4.2.7 Xforms

Am 14. Oktober 2003 gab das World Wide Web Consortium (W3C) die *Xforms 1.0 Recommendation* frei. Xforms 1.0 eine neue Generation von Internet-basierten Formularen begründen, welche im Gegensatz zu HTML-Formularen zwischen Zweck, Präsentation und Ergebnis mit Hilfe der Extensible Markup Language trennen können [Xfor03].

Die Gestaltung von Xforms beruht auf der Analyse der Nutzung von HTML-Formularen etwa in den vergangenen Zehn Jahren, um Formulare zu verbessern. Xforms erlaubt somit alles, was HTML-Formulare auch bieten und darüber hinaus die folgenden Fähigkeiten:

- Eine Format Kontrolle erlaubt die Prüfung von Daten noch während der Eingabe.
- Die Felder von Daten, ohne die ein Formular ungültig wäre, können gekennzeichnet werden.
- Es können Bereichswerte für Datenfelder angegeben werden.
- Formulardaten können in XML kodiert übertragen werden.
- Die Integration von WWW-Diensten, wie etwa SOAP oder XML RPC ist möglich.
- Die selben Formular-Daten können zu mehreren Servern gleichzeitig übertragen werden.
- Formulare können in eine Datei gespeichert und nachher wieder geladen werden.
- Eingabedaten eines Formulars können als Basis für folgende Formulare verwendet werden.
- Werte eines Formulars können aus anderen Werten berechnet werden.
- Der Aufbau eines ‚Warenkorbs' oder eines ‚Assistenten' kann ohne Skript realisiert werden.

4.2.7.1 Eigenschaften von Xforms

- *Xforms verbessert die Benutzbarkeit:*
  Xforms wurde so gestaltet, dass der verwendete Internet-Browser die Eingaben aktiv

unterstützen kann. So kann während einer Eingabe geprüft werden, ob die Daten einem gewünschten Format entsprechen, oder ob die Daten gültige Eingaben darstellen. Beispiel: Ein Anfangstermin muss vor einem Endtermin beginnen. Dies reduziert den Kommunikationsaufwand zwischen WWW-Server und Internet-Browser und verringert dadurch die Antwortzeiten bei der Eingabe. Der Benutzer erhält eine unmittelbare Rückmeldung (Feedback) vom System anstelle von Netz- bzw. Lastbedingten, ‚hängenden' Verzögerungen.

- *Xforms erleichtert die Erstellung von Formularen:*
  Xforms erlaubt die Deklaration von Eigenschaften für Eingabewerte und die Beschreibung von Abhängigkeiten zwischen Werten. Der Autor kann ohne die Verwendung von Skripten adaptive Formulare erstellen. Deshalb können selbst umfangreiche Formulare adaptiv ausgelegt werden, ohne die Erstellung kompliziert werden zu lassen. Im Gegensatz zu HTML-Formularen sind Xforms Formulare einfacher und weniger umfangreich.

- *Xforms ist in XML kodiert und benutzt XML für die Datenübermittlung:*
  Jedes Xforms-Dokument ist XML-konform kodiert und liefert XML kodierte Ausgaben zurück. Auch können XML konforme Daten als Eingabe für Xforms-Dokumente genutzt werden. Damit kann die Kette zwischen Benutzer und System komplett im gleichen Datenformat gehalten werden.

- *Xforms greift auf existierende XML Konzepte zurück:*
  Xforms benutzt existierende XML-Techniken wie etwa *Xpath* zur Adressierung und Berechnung von Werten, und das XML-Schema zur Definition von Datentypen. Daraus resultiert ein doppelter Nutzen: Zum einen kann Xforms leichter von Personen erlernt werden, die XML schon kennen und zum anderen können Autoren die verfügbaren XML-Editoren für die Erstellung ihrer Xforms Dokumente benutzen.

- *Xforms ist Geräteunabhängig:*
  Ein Xforms Dokument kann auf viele verschiedene Endgeräte transportiert und dort mit Xforms kompatiblen Browsern dargestellt werden. Dies entspricht der im Rahmen dieser Arbeit geforderten Abstraktion der Interaktion von spezifischen Endgeräten!

- *Xforms ist internationalisiert:*
  Weil der Datenaustausch bei der Verarbeitung eines Xforms Dokuments auf der Basis von XML stattfindet, ist Xforms international einsetzbar.

- *Xforms anpassbar für behinderte Menschen:*
  Xforms ist so gestaltet, dass es an spezielle Bedürfnisse (z.B. an die von behinderten Menschen) angepasst werden kann.

4.2.7.2 Verfügbare Implementationen

Die *Xforms Working Group* welche den Xforms Standard entwickelt, setzt sich zusammen aus Mitgliedern der führenden Firmen aus dem IT Bereich. Daraus sind eine ganze Reihe verfügbarer Implementierungen hervorgegangen. Tatsächlich stellt Xforms z. Zt. einen der meist implementierten W3C Standards dar.

Einige Entwicklungen resultierten in einer Xforms Erweiterung für die Verwendung mit Internet-Browsern. Andere Umsetzungen realisieren eine Transformation vorhandener HTML-Formulare zu Xforms kompatiblen Formularen auf dem HTML-Server. Damit können alte HTML-Formulare mit neueren Browsern genutzt werden, die Xforms bereits interpretieren können.

4.2.7.3 Klassifikation

Xform kann bedingt als eine modellbasierte Beschreibungssprache verstanden werden, die auf der Basis eines Dokumenten- bzw. Formularmodells arbeitet. Der Abstraktionsgrad bewegt sich jedoch wenig über der Toolkit-Ebene.

Xforms ist eine Weiterentwicklung der bekannten HTML-Formulare. Bestimmte wiederkehrende Problemstellungen, die mit HTML-Formularen umständlich und nur durch zu Hilfenahme von Skriptsprachen realisiert werden können sind mit Xforms einfach und ohne Skript machbar. Es finden sich Funktionen zur Formatprüfung der eingegebenen Daten und es können Abhängigkeiten zwischen verschiedenen Daten Feldern definiert werden. Weil der Internet-Browser die Bearbeitung der Funktionen erledigt und kein Rückgriff auf den Server nötig ist werden Latenzzeiten bei der Interaktion eliminiert, wie sie häufig bei HTML-Formularen auftreten.

4.3 Systeme

Im vorhergehenden Abschnitt wurden Beschreibungssprachen und ihre Wirkungsweisen dargestellt. In diesem Abschnitt werden aktuelle Systeme betrachtet.

4.3.1 Ubiquitous Interactor (UBI)

Der Ubiquitous Interactor (UBI) [NBW05] wurde im Rahmen des sView-Projekts am Swedish Institute of Computer Science entwickelt [BE01]. Dem System liegt die Sichtweise zugrunde, dass Anwendungsprogramme nicht fest installiert auf einem Gerät vorliegen, sondern vielmehr persönliche, geräteunabhängige Dienste darstellen. Das sView-System stellt eine personalisierte Umgebung für solche elektronischen Dienste bereit. Auf den Endgeräten (PC oder PDA) des Benutzers wird ein *sView User Server* installiert, der den sog. *Briefcase* des Benutzers bearbeitet. Der Briefcase enthält alle personalisierten Dienste des Benutzers und kann durch einen PC/PDA per Netzwerk oder durch ein Mobiltelefon mittels WAP zugegriffen werden. Ein Briefcase wird durch den *sView Enterprise Server* bereitgestellt, der auf einem zentralen (Firmen-)Rechner installiert ist. Die eigentliche Ausführung der Dienste kann je nach Typ auch ohne stehende Netzwerkverbindung erfolgen (Offline-Betrieb).

4.3.1.1 Aufbau

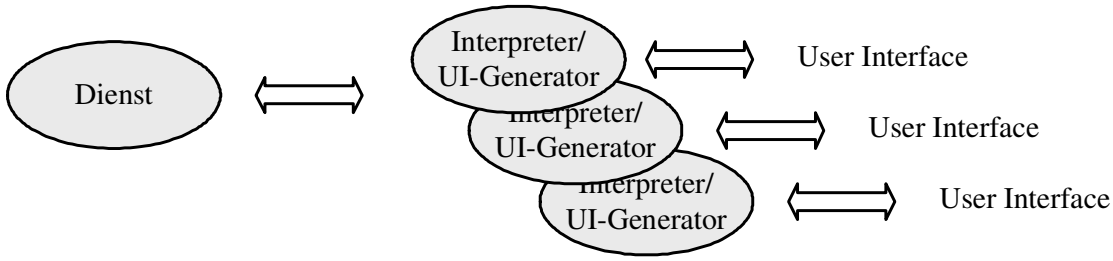

Abbildung 38: UBI erzeugt Benutzungsoberflächen durch Interpreter nach [NBW05]

Die im sView System bereitgestellten personalisierten Dienste entsprechen den mobilen Agenten dieser Arbeit. Die Dienste des sView-Systems können durch einen sog. *Ubiquitous Interactor* (UBI) mit dem Benutzer interagieren. Dazu verfügt jeder Dienst über eine Dialogbeschreibung, welche die Interaktion auf der Ebene von Interaktionsschritten beschreibt. Die Dialogbeschreibung wird im UBI von einer Anzahl von UI-Generatoren, die als Interpreter arbeiten, erzeugt und bearbeitet. Jeder Interpreter ist für einen bestimmten Endgerätetyp realisiert und erzeugt für das Gerät eine Benutzungsoberfläche (Siehe Abbildung 38).

4.3.1.2 Beschreibungssprache

Die XML-kompatible Beschreibungssprache ISL[13] führt die abstrakte Beschreibung von Benutzungsoberflächen auf so genannte *Acts* zurück. Acts können als abstrakte Handlungsanweisungen verstanden werden, welche zur Erreichung von Eingaben durchzuführen sind. UBI unterscheidet insgesamt 8 verschiedene Arten von Acts*: Input, Output, Select, Modify, Create, Destroy, Start* und *Stop*.

- Ein *Input*-Act bietet dem Benutzer eine Eingabemöglichkeit an.
- Ein *Output*-Act führt eine Ausgabe an den Benutzer durch.
- Ein *Select*-Act veranlasst den Benutzer eines aus mehreren Elementen auszuwählen.
- Ein *Modify*-Act erlaubt die Änderung von Daten im System.
- Mit einem *Destroy*-Act können Daten im System gelöscht werden.
- *Start* und *Stop* beginnen und beenden die Interaktion mit dem Benutzer.

Die Acts besitzen einige Attribute, die mit ISL verändert werden können. Darunter finden sich Bezeichner zur eindeutigen Unterscheidung der Acts. Es ist möglich festzulegen, ob ein Act für eine gewisse Dauer gültig ist (z.B. Splashscreen), ob er modal ist und andere Acts blockiert oder ob er gleichzeitig mit anderen Acts ausgeführt werden kann. Für Input-Acts können Default-

[13] ISL = Interaction Specification Language

werte angegeben werden. Zusätzlich können Metainformationen beigefügt werden, die z.B. über Domäneninformationen verfügen.

4.3.1.3 Präsentation

Eine UBI-Präsentation wird über eine oder mehreren parallel laufende Interaction Engines vorgenommen. Jede Interaction Engine verarbeitet einerseits abstrakte ISL Beschreibungen, andererseits kann sie auch konkrete medienspezifische Inhalte in eine Präsentation integrieren. Die medienspezifischen Inhalte werden separat in einem *Customization Form* abgelegt. Eine Interaction Engine führt die abstrakten Acts mit den Informationen des Customization Form zur Laufzeit zusammen.

Für die UBI-Spezifikation einer Benutzungsoberfläche ergeben sich drei Layer gemäß Abbildung 39.

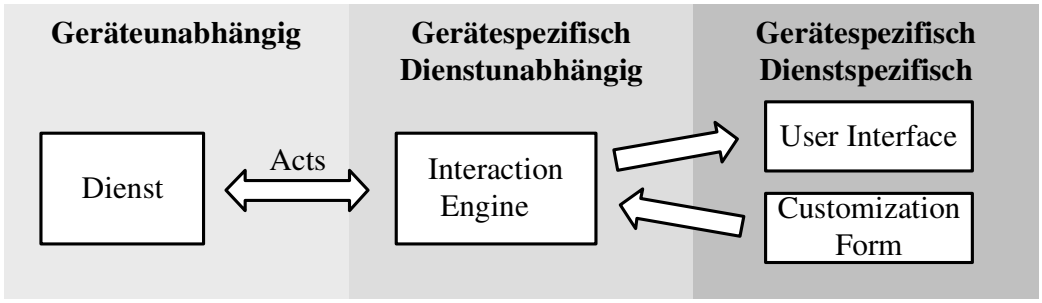

Abbildung 39: Die drei Layer einer UBI-Spezifikation nach [NBW05]

Bei Präsentationsbeginn wird einem Dialog ein systemweit eindeutiger Bezeichner zugewiesen. Ein weiterer wichtiger Parameter ist der *life-cycle*-Wert. Er legt die Dauer eines Dialoges fest und kann die Werte *temporary*, *confirmed* und *persistent* annehmen. Der Wert legt fest, ob die Gültigkeit eines Dialogs nach einiger Zeit, nach einer bestimmten Eingabe oder niemals erlischt.

4.3.1.4 Klassifikation

Der Ubiquitous Interactor verwendet als Basis zur Erzeugung von Benutzungsoberflächen ein abstraktes Dialogmodell und kann damit den modellbasierten UIMS zugeordnet werden. Die Beschreibungssprache ISL erlaubt die abstrakte Kodierung von Dialogen in Form von interaktiven Handlungsschritten.

Die Acts werden mit Hilfe von Interaction Engines interpretiert und zu konkret darstellbaren Elementen umgesetzt. Eine Zugabe von medienspezifischen Inhalten wird durch das Customization Form erreicht.

Aus den Publikationen zum UBI (z.B. [NBW05]oder [NB03]) geht kein Hinweis darauf hervor, dass es eine Synchronisation zwischen den Interaction Engines vorgenommen wird. Als Folge

können parallel laufende Interaction Engines den Interaktionsprozess nur bedingt oder gar nicht untereinander koordinieren.

4.3.2 ITS

Das IST-System ist ein modellbasiertes Entwicklungswerkzeug, das 1990 im IBM T.J. Watson Research Center entwickelt wurde [WBBG+90]. Das Ziel des Systems war die Bereitstellung eines Entwicklungswerkzeugs für Anwendungsprogramme, die sich automatisch unterschiedlichen Bedingungen auf den Endgeräten anpassen können. Die Programme sollten mit verschiedenen grafischen Fähigkeiten (Auflösung, Farbtiefe), verschiedenen Eingabetechniken, und der Sprache des Benutzers umgehen können.

4.3.2.1 Fähigkeiten

Das IST-System zielte auf die Generierung von grafischen Benutzungsoberflächen ab. Dazu wurde eine ITS-Applikation in vier Ebenen eingeteilt: *Aktion*, *Dialog*, *Gestaltungsregeln* und *Gestaltungsprogramme*[14].

Die Ebenen erledigen die folgenden Aufgaben:

- Die *Aktions*-Ebene behandelt das Schreiben und Lesen von Daten,
- die *Dialog*-Ebene behandelt die Abfolge der Interaktion,
- die *Gestaltungsregeln* spezifizieren die Darstellung der Dialogelemente und
- die *Gestaltungsprogramme* bringen die Dialogelemente zur Darstellung.

Das System war so ausgelegt, dass Nicht-Programmierer in die Lage versetzt wurden Benutzungsoberflächen zu erzeugen. Das System adressierte grafische Benutzungsoberflächen für Desktop-Anwendungen. Mobile Endgeräte wurden nicht in Betracht gezogen.

4.3.2.2 Klassifikation

Das ITS-System ist ein modellbasiertes UIMS auf der Basis eines Dialogmodells. Es ist auf Benutzungsoberflächen für Desktop-Systeme ausgerichtet. Deshalb konzentriert sich das ITS-System auf Problematiken wie etwa die Anpassung des Farbraums oder auf die Berücksichtigung unterschiedlich großer Bildschirme. Andere Interaktionsmethoden, wie sie bei mobilen Endgeräten zu finden sind, kamen bei ITS noch nicht in Betracht.

Ein Manko des ITS-Systems ist die Art und Weise, wie die Gestaltungsregeln eingesetzt werden. Es wurde bei der Entwicklung von ITS davon ausgegangen, dass die Gestaltungsregeln eines Anwendungsprogramms nicht auf andere Programme übertragen werden können. Dafür wurden keine zufriedenstellenden Ergebnisse vorhergesehen. Infolgedessen schlossen die ITS-

[14] Im Original. Action, Dialog, Style Rules und Style Programs

Entwickler darauf, dass Anwendungsprogramme möglichst keine spezifischen Gestaltungsmerkmale aufweisen sollten.

4.3.3 MUSA

Das MUSA-System ist ein prototypisches System, das im Rahmen von Forschungsarbeiten im Software Research Lab der Universität Salzburg entstanden ist. Das *MUSA-System* implementiert das *Multi User Interface Single Applicationn Model* (MUSA-Modell) nach Fischmeister, Menkhaus und Pree [FMP02]. Das MUSA-System unterstützt die Adaption von Benutzungsoberflächen an die Konfiguration des Benutzers und die Bildauflösung. Zur abstrakten Beschreibung der Benutzungsoberflächen wird das proprietäre EGXML verwendet.

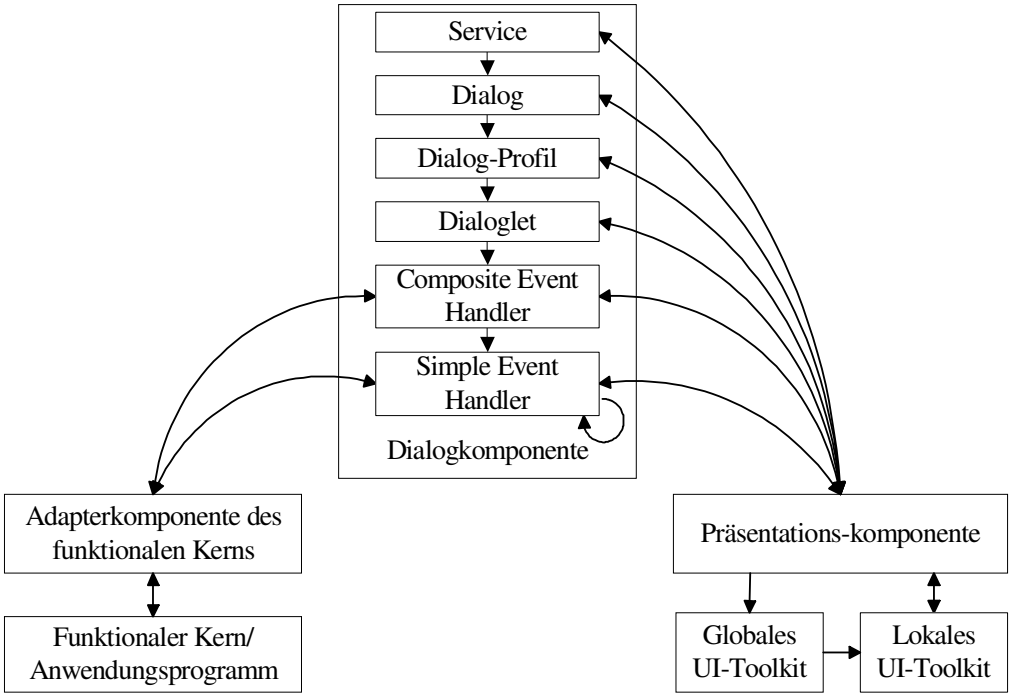

Abbildung 40: Das Musa Modell nach [FMP02]

MUSA ist eine Variante des Presentation-Abstraction-Controller (PAC)-Amodeus Modells [CCN97] und unterteilt gemäß Abbildung 40 die Architektur eines Systems mit Benutzungsschnittstelle in Präsentations-, Dialog- und Anwendungskomponente.

Die Dialogkomponente des MUSA-Systems ist als sog. Event Handler Graph realisiert. Der *Event Handler-Graph* vermittelt zwischen der Präsentationskomponente an der Benutzungs-

schnittstelle und der Anwendung. Er verarbeitet Ereignisse, die der Benutzer mittels der Benutzungsoberfläche an die Anwendung schickt.

4.3.3.1 Event Handler Graph

Der Event Handler-Graph (EH-Graph) ist die zentrale Komponente des MUSA-Systems. Der EH-Graph ist eine Repräsentationsform eines Dialogs. Aus einem EH-Graphen können an bestimmte Bildgrößen angepasste Dialoge erzeugt werden. Dieser Vorgang wird als *Tailoring* bezeichnet. Das Tailoring arbeitet ähnlich dem bei IBM-Websphere angewendeten *Fragmentation* Prozess [Brit01]. Weil die Anzeigeflächen von mobilen Endgeräten geringere Auflösungen als Desktop Systeme besitzen, die Dialoge aber sowohl auf Desktop- als auch auf mobilen Systemen dargestellt werden sollen, ist diese Anpassung der Dialoge erforderlich. Ein EH-Graph wird auf Basis einer abstrakten Beschreibung in EGXML vom MUSA-System erzeugt [FM02].

4.3.3.2 Adaption der Darstellungsfläche

Der Adaptionsprozess wird in MUSA auch als *Geräteunabhängiger Autorenprozess* bezeichnet. (Siehe Abbildung 41).

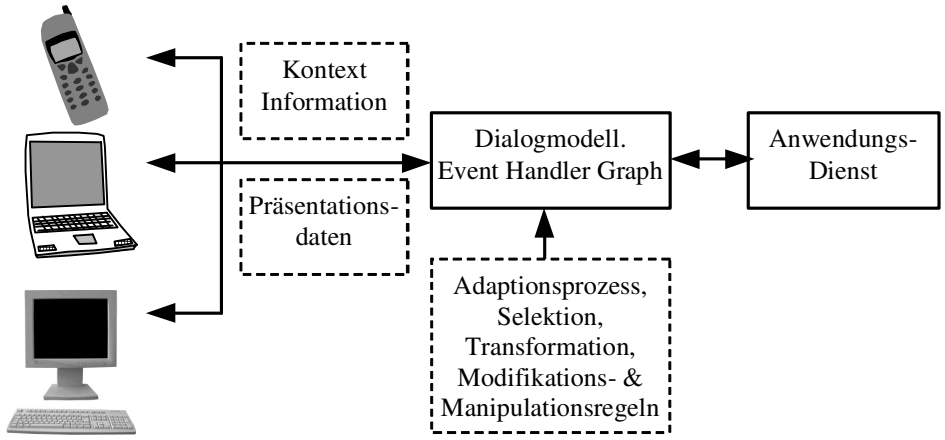

Abbildung 41: Der geräteunabhängige Autorenprozess [MF02]

Im MUSA System ist jeder Dialog in Knoten zerlegt, die als Event Handler implementiert sind.Weil der Graph eine Struktur vorgibt, kann das System ein *adaptives Clustering* dieser Knoten vornehmen. Damit ist die Zusammenfassung mehrerer Knoten zu einem Meta-Knoten gemeint. Auf diese Weise entsteht eine hierarchische Graphenstruktur. Das MUSA-System benutzt diese Struktur, um die Dialogelemente für den Benutzer erkennbar zu ordnen. Dieses Verfahren erlaubt die stufenweise Adaption. Ist wenig Fläche zur Präsentation vorhanden, kann die Darstellung auf ein Element, das einem Dialogschritt entspricht und einige Navigationselemente reduziert werden. Wenn die Darstellungsfläche vergrößert wird, können weitere Dialogelemente hinzugenommen werden.

4.3.3.3 Klassifikation

Das MUSA-System kann den modellbasierten UIMS zugeordnet werden. Benutzungsoberflächen werden auf der Basis eines Dialogmodells beschrieben, das auf verschiedene Endgeräte abbildbar ist. Das Dialogmodell wird als EH-Graph bezeichnet. Die Beschreibung der EH-Graphen erfolgt in der XML kompatiblen EGXML.

MUSA konzentriert sich darauf, die Dialogelemente einer Benutzungsoberfläche mittels des Event Handler Graphen in sinnvolle Teile zu zerlegen, bzw. bereits bei der Dialogerstellung eine strukturierte Dialogerstellung zu fördern. Das wirkt sich besonders bei kleinen Endgeräten vorteilhaft aus, denn es gelingt damit die Reduktion der gleichzeitig darzustellenden Dialogobjekte ohne den Verlust von strukturierter Dialogkontrolle.

4.3.4 Microsoft .NET Framework

Microsoft .NET bezeichnet eine Entwicklungsplattform für Internetdienste. Die Plattform stellt eine Laufzeitumgebung bereit, die den Anwendungsprogrammen, Web-Diensten und Geräten die Kooperation ermöglicht [Micro02a]. Die Zielsysteme der Plattform sind z.B. Desktop-PCs, Laptop-PCs, PDAs und Mobiltelefone [Micro02b]. Voraussetzung für Installation und Betrieb von Microsoft .NET ist eines der folgenden Betriebsysteme: Microsoft Windows XP, Windows XP Embedded oder PocketCE .NET, denn Microsoft .NET auf Komponenten greift auf Komponenten der Microsoft-Betriebsysteme zu.

Das Windows .NET Framework besteht aus der Common Language Runtime (CLR) und einer Reihe einheitlicher Klassenbibliotheken. Die CLR ist für Laufzeitdienste zuständig, wie z. B. die Sprachintegration, Durchsetzung von Sicherheitsrichtlinien und die Speicher-, Prozess- und Threadverwaltung. Die Klassenbibliotheken bieten Standardfunktionen, z. B. Eingabe/Ausgabe, Zeichenfolgenbearbeitung, Sicherheitsverwaltung, Netzwerkkommunikation, Threadverwaltung, Textverwaltung und Funktionen für den Entwurf von Benutzungsoberflächen. U.A wird auch der Zugriff auf OLE DB-, ODBC-, Oracle- SQL Server-Schnittstellen ermöglicht.

Für die Entwicklung von Netzanwendungen stehen dem Entwickler die Anwendungstypen *Web-Formular* und *Windows-Formular* zur Verfügung:

- *Web-Formulare*
  Ein Web-Formular bezeichnet einen serverbasierten Ansatz, wie er durch HTML-Formulare oder XFORMS (vgl. Abschnitt 4.2.7) gegeben ist. Ein Web-Formular besteht somit aus einer Markup-Sprache, die der Kodierung einer Benutzungsoberfläche dient. Dazu kommt Programmcode, der den funktionalen Kern realisiert. Der Programmcode kann in einer von der CLR unterstützten Programmiersprache realisiert sein, wie etwa: C#, Visual Basic, Jscript.NET, etc… Für die Unterstützung der Web-Formulare durch mobile Endgeräte wird das *Microsoft Mobile Internet Toolkit* (MMIT) angeboten.
- *Windows-Formulare*
  Ein Windows-Formular setzt sich aus denselben Klassen zusammen wie die Anwendungs-

programme der *Microsoft-Windows*-Plattform. Ein Entwickler, der mit den Microsoft-Klassen bereits vertraut ist, profitiert demzufolge von seiner Erfahrung. Das *.NET Compact Framework* erlaubt den Betrieb der Windows-Formulare auf mobilen Endgeräten.

Diese beiden Anwendungstypen werden im Folgenden näher betrachtet.

4.3.4.1 Microsoft Mobile Internet Toolkit

Das Microsoft Mobile Internet Toolkit (MMIT) bezeichnet ein serverbasiertes System zur Bereitstellung von Netzanwendungen für mobile Endgeräte. Das MMIT stellt mit der *Mobile Internet Control Runtime* eine Laufzeitumgebung bereit, die auf den mobilen Geräten installiert wird. Die Laufzeitumgebung hält ihrerseits verschiedene *MMIT Runtime Rendering Modules* für die Präsentation von Benutzungsoberflächen bereit. Die Runtime Rendering Modules übernehmen die Anpassung der Interaktionsdaten an die Endgeräteschnittstellen.

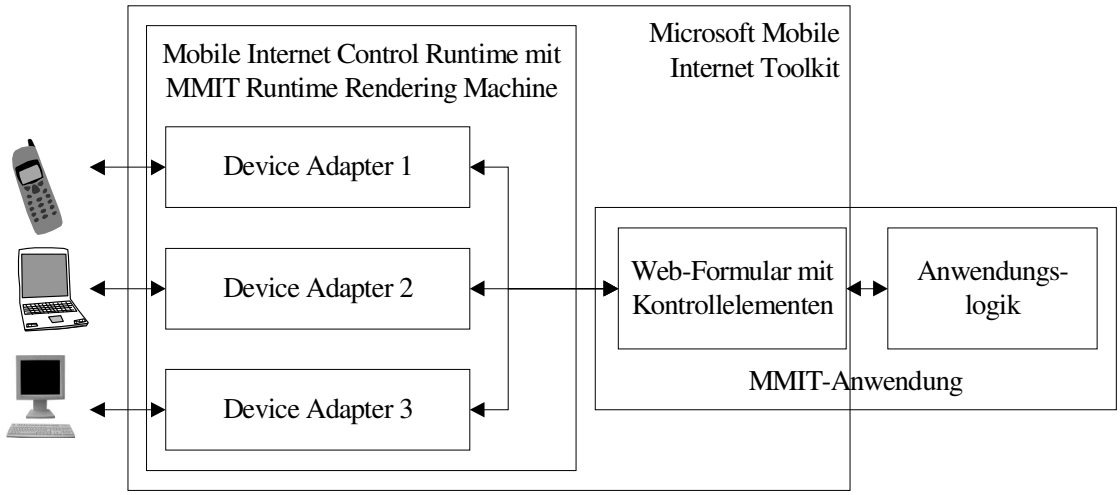

Abbildung 42: Microsoft Mobile Internet Toolkit

Entsprechend obiger Abbildung 42 besteht eine MMIT-Anwendung aus einem Web-Formular mit Kontrollelementen und aus der Anwendungslogik. Die Kontrollelemente stellen abstrakte Elemente der Benutzungsoberfläche dar, die von einem *Device Adapter* in konkrete Elemente umgesetzt werden. Ein Kontrollelement des Web-Formulars kann vom Device Adapter auf typische Elemente z.B. eines HTML-Formulars abgebildet werden. Der Entwickler muss sich daher nicht weiter mit der konkreten der Technologie auf dem Frontend befassen. Daher ist die Integration der durch .NET unterstützten Frontends leicht. Das Austauschformat der Daten auf dem Frontend ist unerheblich für das Web-Formular, da es von Render Adapter für den Entwickler transparent angepasst wird.

Ein Schlüsselkonzept des MMIT ist seine Fähigkeit ein Web-Formular für die Bildauflösung eines Zielgeräts anzupassen. Dabei werden die Verfahren *Pagination* und *Chunking* angewendet.

- *Pagination*
  beschreibt ein Verfahren die Elemente eines Formulars in eine Sequenz kleiner Formulare umzusetzen, ähnlich dem Verfahren aus Abschnitt 5.7. Im Gegensatz zum Verfahren des MMIT, das eine lineare Liste aus Formularen generiert, bleibt beim Verfahren aus 5.7 die (beliebige) Dialogstruktur erhalten.
- *Chunking*
  wird bei der Präsentation von Listenelementen angewendet. Das Listenelement präsentiert dem Benutzer eine Anzahl von Einträgen. Übersteigt die Zahl der Einträge einen gewissen Wert, wird die Liste in eine Sequenz von Formularen zerlegt. Das in dieser Arbeit entwickelte Konzept sieht ein ähnliches Verfahren vor, das beim graphischen Rendermodul zum Einsatz kommt (vgl. Abschnitt 6.10.2).

Das MMIT erlaubt durch diese Verfahren eine gewisse Adaption an kleine mobile Endgeräte. Die bei der Adaption angewendeten Verfahren beziehen sich primär auf grafische Interaktionsschnittstellen. Das Chunking Verfahren erlaubt die Transformation eines Formulars in eine Anzahl neuer Formulare, die in einer Folge von Formularen dargestellt werden. Dies kann die Bedienung insbesondere von Formularen mit vielen Bedienelementen für den Endanwender unübersichtlich werden lassen, da die Bedienelemente in sehr vielen Formularen resultieren. Einzelne Formulare sind durch die einfache Navigation schlecht zu erreichen, denn es werden dem Benutzer lediglich die Kommandos *Vor* und *Zurück* angeboten. Auf sprachliche Benutzungsoberflächen sind diese Verfahren ebenfalls nicht speziell ausgelegt.

Ein Nachteil der Web-Formulare besteht in der Natur des Ansatzes, einen Server im Netz zu verwenden und das Anwendungsprogramm dort zu betreiben, während auf dem Frontend-Gerät die Präsentation vorgenommen wird. Diese Variante kann nur bei ständiger Verfügbarkeit des Zugriffs zum Server seine Dienste anbieten. Wird die Verbindung unterbrochen, ist auch die Verbindung zum funktionalen Kern der Anwendung getrennt, wodurch die Fortführung der Arbeit mit einer Anwendung gestört ist.

4.3.4.2 .NET Compact Framework

Die Anwendungen für das .NET Compact Framework werden als *Windows-Formulare* bezeichnet. Die Windows-Formulare stehen im Gegensatz zu den Web-Formularen, die dem Gebrauch eines Internet-Browsers gleichen, der HTML-Dokumente lädt. Windows-Formulare werden auf dem Endgerät selbst ausgeführt.

Das .NET Compact Framework ist eine eingeschränkte Version des .NET-Frameworks, das zum Ausführen der .NET Anwendungen benötigt wird. Es ermöglicht Entwicklern jederzeit und an jedem Ort Codes oder Webservices mit Hilfe von PDAs oder Handys abzurufen.

Der Entwickler einer mobilen Anwendung profitiert von Kenntnissen, die er bei der Realisierung von Software für stationäre Microsoft Betriebssysteme erworben hat. Weil das .NET Compact Framework ein Subset der stationären Umgebung darstellt, ist er schnell in der Lage, Anwendungsprogramme für mobile Engeräte zu realisieren. Die Portierung einer desktopbasierten Windows-Anwendung zu einer .NET Compact Framework Anwendung kann somit einfacher vorgenommen werden.

Die Performanz der .NET Compact Framework Anwendung wird bei komplexeren Algorithmen gegenüber einem Web-Formular weitaus höher ausfallen, da keine Interpretation einer Metasprache vorgenommen wird, sondern die Ausführung in optimierten Programm-Modulen in der jeweiligen Maschinensprache der CPU erfolgen kann.

Die Ausführung der .NET Compact Framework Anwendungen bedingt die Installation der .NET Plattform auf den Endgeräten. Diese ist jedoch für viele mobile Geräte nicht verfügbar. Aktuell wird .NET weder durch das im Bereich der mobilen Telekommunikation sehr verbreitete Betriebssystem *Symbian*, noch durch das PDA Betriebsystem *PalmOS* unterstützt. Dadurch sind Nutzer dieser Geräte vom Gebrauch der .NET Anwendungen ausgenommen.

Abbildung 43: Eine .NET-Spieleanwendung auf dem PocketPC-Emulator

Obige Abbildung 43 zeigt eine Beispielanwendung mit dem .NET Compact Framework. Das .NET Compact Framework verwendet keine speziellen Adaptionsverfahren für Benutzungsoberflächen. Die Anpassung erfolgt durch die Skalierung der Widgets, was besonders bei sehr kleinen Anzeigeflächen problematisch ist.

4.3.4.3 Klassifikation

Die Behandlung der Benutzungsoberflächen bei den Windows-Formularen ist toolkitbasiert. Ein Indiz hierfür ist die Anpassung der Benutzungsoberflächen auf Toolkit-Ebene durch die Skalierung der Bedienelemente.

Web-Formulare bewegen sich hingegen an der Grenze zu einem modellbasierten Ansatz, auf der Basis eines Dialogmodells. Jedoch ist der Abstraktionsgrad der Benutzungsoberfläche äußerst gering.

4.3.5 Visual Obliq

Das Visual Obliq-Projekt, dessen Ergebnis u. A. das Visual Obliq-System darstellt, hatte die Entwicklung von verteilten Anwendungsprogrammen zum Inhalt. Es war Gegenstand der Forschungsarbeiten im *Systems Research Center* der Fa. *Digital*. Die Zielsetzung des Visual Obliq-Projekts war es, die Programmierung einer verteilten Anwendung so einfach zu machen wie die Erstellung von Standardanwendungen [BB94]. Die Visual Obliq-Entwicklungsumgebung besteht daher aus einem interaktiven Werkzeug zur visuellen Anwendungserstellung. Dazu gibt es eine Laufzeitumgebung für die Ausführung von mobilem Programmkode. Die Laufzeitumgebung ermöglicht die Migration der Programme auf andere Rechner während der Laufzeit. Auf der Basis von Visual Obliq beschreiben Bharat und Cardelli in [BC95] ein System, welches mobile Anwendungsprogramme realisiert.

4.3.5.1 Aufbau

Das Visual Obliq-System stellt eine Laufzeitumgebung für mobile und verteilte Anwendungsprogramme bereit. Programme können auf einem Rechnersystem gestartet werden und innerhalb ihrer Laufzeit auf einen anderen Rechner springen, um dort die Programmausführung fort zu führen. Um dies tun zu können, wird bei einer Migration der aktuelle Programmstatus ermittelt und ebenfalls auf das Zielsystem übertragen. Vor Programmausführung auf dem Zielsystem kann damit der alte Programmkontext wieder hergestellt werden.

Im Visual Obliq-System werden Programme als Datenstrukturen behandelt, die sich aus veränderlichen und aus unveränderlichen Daten zusammensetzen. Die Datenstrukturen weisen eine hierarchische Struktur auf und können als Graph dargestellt werden. Bei der Migration eines Programms auf ein anderes System werden nun immer nur die unveränderlichen Daten an das Zielsystem übermittelt. An Stelle der veränderlichen Daten werden Referenzen benutzt.

Abbildung 44 zeigt die Migration eines Datengraphen. Die hervorgehobenen Elemente stellen veränderliche Datenelemente dar, die nicht als Kopie, sondern als Referenz übertragen werden. Die Migration von Ausgangs- nach Zielsystem ist daher nicht vollständig. Dies hat zur Folge, dass zur Ausführung der mobilen Programme immer eine Online-Verbindung erforderlich ist. Neben dieser Form der Migration kann alternativ auch die Anfertigung einer Kopie eines Datengraphen angefordert werden. Dann ist die Online-Verbindung nicht zur weiteren Ausführung nötig. Diese Vorgehensweise ist die Basis für die in [BC95] beschriebenen mobilen Agenten, die sich innerhalb des Verbundes von Visual Obliq-Systemen im Netzwerk bewegen können.

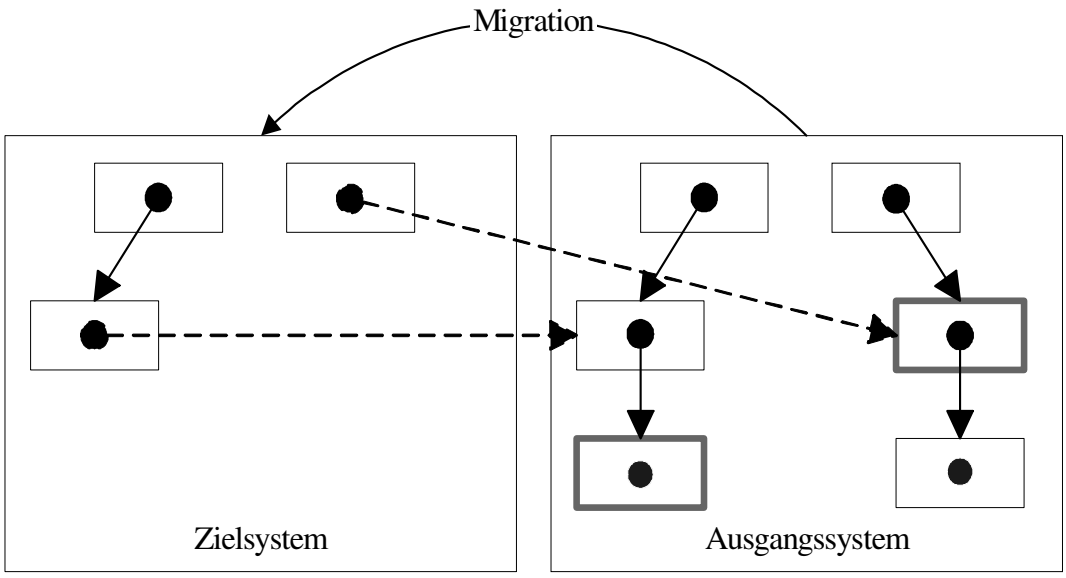

Abbildung 44: Migration eines Datengraphen nach [BC95]

Um die Migration durchzuführen, muss der Agent einen *hop-Befehl* ausführen. Vor der Migration wird dem Agenten dann eine *Suitcase-Datenstruktur* übergeben. Diese Struktur enthält Daten, die vom Zielsystem gebraucht werden, um den Programmkontext wieder herzustellen. Nach der Migration bekommt der Agent eine sog. *Briefing-Datenstruktur*, die ihm Informationen über den lokalen Kontext gibt.

Analog zur Unterstützung der mobilen und kontinuierlichen Programmausführung wird auch die Benutzungsoberfläche auf dem Zielsystem wieder hergestellt. Vor der Abreise eines Agenten wird der Zustand der Benutzungsoberfläche ermittelt, indem Anordnung und Typ der Bedienelemente und der dazugehörige Status aller Elemente ermittelt wird. Der Status wird ebenfalls in der S*uitcase-Datenstruktur* gespeichert und auf das Zielsystem übertragen. Auf dem Zielsystem kann so die Benutzungsoberfläche identisch rekonstruiert werden.

4.3.5.2 Klassifikation

Das Visual Obliq-System ist toolkitbasiert. Dementsprechend niedrig ist der Abstraktionsgrad der Interaktionsdaten. Die Adaption verschiedener Endgeräte ist in diesem System nur ansatzweise möglich.

Das Visual Obliq-System ermöglicht die Migration von Anwendungsprogrammen auf verschiedene Rechnersysteme. Die Zustandserhaltung der Dialoge erfolgt durch die Abfrage der Dialogobjekte auf der Toolkit-Ebene. Auf dem Zielsystem kann eine mobile Anwendung die Widgets seiner Benutzungsoberfläche neu generieren und sie dann mit den alten Werten initialisieren.

4.3.6 SmartKom

SmartKom ist ein vom Bundesministerium für Bildung und Forschung (BMBF) gefördertes Projekt. Das Konsortium besteht aus 12 Partnern, unter denen sich bekannte Namen wie Siemens, DaimlerChrysler AG und Phillips finden. Das Projekt startete im September 1999 und endete vier Jahre später im September 2003. In SmartKom wurden Konzepte für neuartige Formen der Mensch-Technik-Interaktion erprobt.

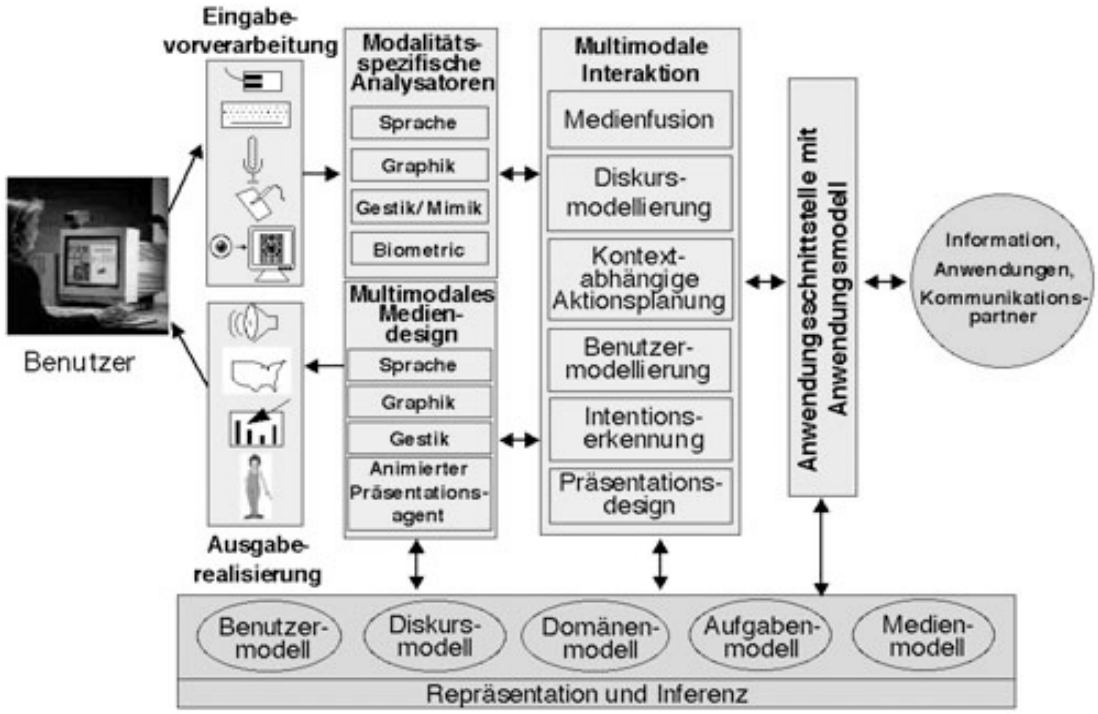

Abbildung 45: Grobarchitektur von SmartKom [Smar03]

Das Projekt ist motiviert durch die steigende Zunahme von IT-Anwendungen in allen Lebensbereichen, die einen Bedarf an effektiveren, effizienteren und natürlicheren Schnittstellen verursacht. Das Ziel von SmartKom ist die Erforschung und Entwicklung einer selbsterklärenden, benutzeradaptiven Schnittstelle für die Mensch-Maschine Interaktion. Ausgangspunkt für die in SmartKom gefundenen Lösungen sind natürliche Dialoge. Deshalb werden verstärkt Interaktionstechniken wie Mimik, Gestik und Sprache in den Realisierungen verwendet.

Dem Projekt liegt die Idee zugrunde, dass eine sprachlich dialogische Interaktion mit graphischen Bedienoberflächen vorteilhaft kombiniert werden kann. Als Resultat soll die

Benutzerschnittstelle höherwertiger sein, da sie die menschlichen Sinne in größerem Umfang als bisher berücksichtigt.

4.3.6.1 Aufbau

Charakteristisch für den in SmartKom verwendeten Ansatz ist, dass Aspekte der *multiplen Codes* und *multiplen Modalitäten* im Zusammenhang mit dialogischer Interaktion im Mittelpunkt stehen.

Der Aufbau des SmartKom-Systems ist in Abbildung 46 dargestellt. Die Realisierung des Systems ist komponentenbasiert. Damit die umfangreichen Funktionen auch auf kleinen Endgeräten zum Einsatz kommen können, ist ein fortwährender Zugang zum SmartKom Kernel notwendig.

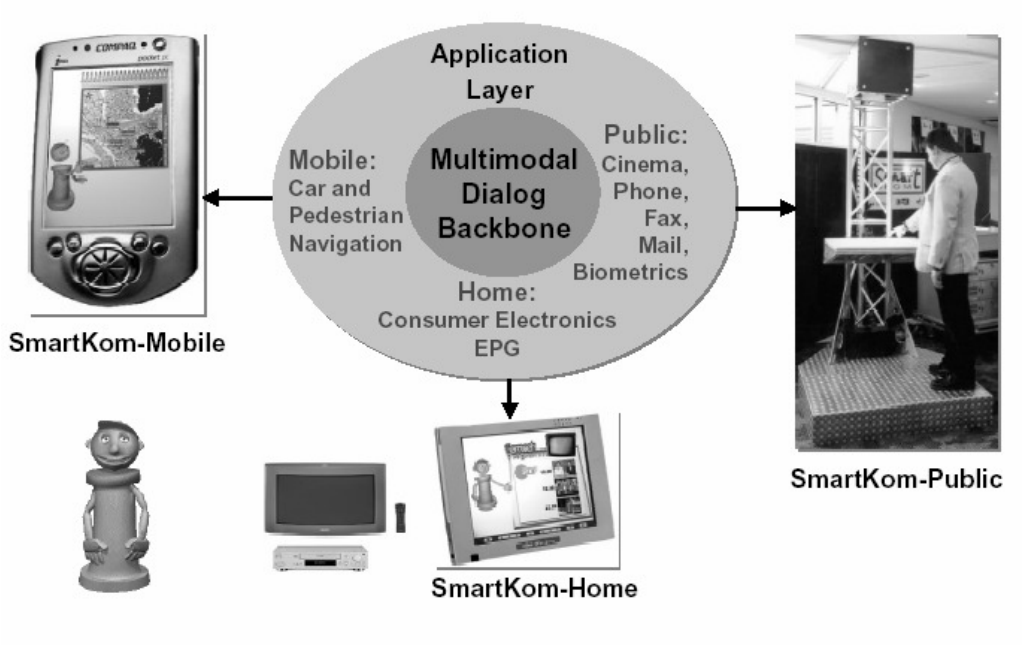

Abbildung 46: SmartKom-Kernel und Anwendungsszenarien [HKMN+03]

SmartKom basiert auf einer verteilten Komponenten-Architektur, die ein sog. *Multi-Blackboard System* realisiert. Die Integrationsplattform MULTIPLATFORM (Multiple Language Target Integration Platform for Modules) dient der Integration der verschiedenen Programmmodule, die in verschiedenen Programmiersprachen implementiert wurden und asynchron zusammenarbeiten [HKMN+03][Wahl03]. Die Plattform enthält eine Virtuelle Maschine als Middleware, welche die Prozesse koordiniert, die auf Rechner im Netz verteilt sind. Die Grobarchitektur in Abbildung 45 zeigt, dass SmartKom eine wissensbasierte Schnittstelle realisiert, die auf Modellen der Benutzer, der Diskursstruktur, der Domäne und der darin gestellten Aufgabe sowie der verfügbaren Medien beruht. Die vier Hauptkomponenten von SmartKom sind die modalitätsspezifischen Analysatoren, die multimodale Interaktionskomponente, die

Anwendungsschnittstelle mit explizitem Anwendungsmodell sowie das multimodale Mediendesign für die Ausgabeplanung.

Wichtiger Teil des SmartKom Systems ist die XML basierte Beschreibung der Interaktionsdaten in der eigens dafür spezifizierten MultiModal Markup Language (M3L). M3L basiert auf ontologischen Beschreibungen und kodiert die Wissensbasis des Systems. Aufgrund des mächtigen Ansatzes ist das System sehr leistungsfähig, es integriert eine breites Spektrum von Technologien: Worterkennung, mimische und gestische Darstellungen. Wie den Online-Präsentationen[15] zu entnehmen ist, gelangen die Assistenzfunktionen so zu großem Ausdruck.

4.3.6.2 Klassifikation

Die Interaktion in SmartKom kann auf ein modellbasiertes UIMS zurückgeführt werden. Den Dialogen liegt ein komplexes Dialogmodell zugrunde, das von einer leistungsfähigen Infrastruktur verarbeitet wird. Die Interaktion konzentriert sich auf assistenzgebende Verfahren.

SmartKom realisiert leistungsfähige Funktionen zur multimodalen Interaktion. Multimodalität bezieht sich hier auf die Einbeziehung möglichst aller zur Verfügung stehenden Interaktionskanäle, um die Qualität der Schnittstelle zu steigern. Die interaktiven Fähigkeiten des Systems haben eine für heutige Verhältnisse sehr hohe Qualität erreicht.

SmartKom ist komponentenbasiert und die Ausführung der äußerst leistungsfähigen Funktionen erfordert eine ebenso leistungsfähige Hardware. Nicht ausreichende Performanz auf dem Endgerät wird durch im Netzwerkverbund befindliche, performante Systeme kompensiert. Die Bearbeitung der ressourcenaufwändigen Funktionen kann bei Bedarf auf diese Systeme verlagert werden.

Die Adaption der Endgeräteschnittstellen erfolgt durch einen multimodalen Ansatz. Von zentraler Bedeutung ist die Dialogführung über Sprache. Grafische Dialoge werden durch das SmartKom-System in die Interaktion einbezogen und können durch Sprache unterstützt bearbeitet werden.

4.3.7 EMBASSI

Das Projekt *Elektronische Multimediale Bedien- und Service Assistenz* – kurz EMBASSI - ist ein vom Bundesministerium für Bildung und Forschung (BMBF) [EMB02] gefördertes Leitprojekt. Es startete im Jahr 1999 und dauerte 4 Jahre. Das Ziel von EMBASSI ist die Entwicklung eines Assistenzkonzepts, das den Nutzer bei der Bedienung von Alltagstechnologie unterstützt. Der Anwender soll mit einem dynamisch vernetzten System arbeiten können, mit dem er auf natürliche Weise interagiert. Durch die Anwendung von multimodalen Verfahren, wie etwa Sprach- und Gestenerkennung, soll er dem System seine Bedürfnisse vermitteln können. Das System soll dabei in der Lage sein, diese Bedürfnisse zu erkennen und zu erfüllen. Die

[15] Auf den Projektseiten (http://www.smartkom.org) finden sich dazu einige Videos. (Stand 1.12.2004)

Infrastruktur wird in EMBASSI durch Alltagsgeräte wie etwa Fernseher, Videorekorder, Autoradios, aber auch durch Computer, etc. gebildet.

4.3.7.1 Einsatzgebiete

Das Projekt zielt auf die folgenden Infrastrukturen:

- *Privathaushalte*
  Bedienassistenz für die Steuerung vernetzter Multimedia- und Infotainment-Systeme.
- *Kraftfahrzeug*
  Fahrerassistenz und Bedienassistenz für die Steuerung von Infotainment-Systemen.
- *Terminalsysteme*
  Bedienassistenz für die Interaktion mit Terminalsystemen wie Bank-, Fahrkartenautomaten.
- *Assistenzsysteme*
  Entwicklerassistenz (Meta-Assistenz) für die Realisierung von multimodalen Assistenzsystemen.

4.3.7.2 Ziele

EMBASSI soll beliebige technische Infrastrukturen zu unterstützen, die vom Endnutzer aus einzelnen Komponenten zusammengestellt werden. Solche Komponenten sind etwa Alltagsgeräte verschiedener Hersteller, die miteinander verbunden werden, wobei jedes Gerät auch einzeln benutzbar sein soll.

EMBASSI soll

- die Unabhängigkeit von Komponenten sichern,
- die dynamische Erweiterbarkeit mit neuen Komponenten ermöglichen,
- zentrale Komponenten vermeiden (einzelne Fehlerquellen, Engpässe),
- verteilte Implementierung unterstützen,
- die flexible Wieder-Nutzung von Komponenten erlauben und
- die Austauschbarkeit von Komponenten ermöglichen.

Die Interaktionsschnittstelle soll auch einen *Assistenzcharakter* aufweisen, dazu werden mehrere Interaktionskanäle bedient:

- Spracheingabe und Sprachausgabe,
- Videobasierte Interaktion (z.B. Gesten- und Positionserkennung),
- Avatare als anthropomorphe Interaktions-Metapher,
- haptisches Feedback an Druck- und Drehknöpfen (z.B. im Umfeld Auto),
- Eingabegeräte, die anthropomorphe Eingabearten ermöglichen.

In EMBASSI wird Multimodalität dazu benutzt, um Modularität und Erweiterbarkeit herzustellen. EMBASSI soll eine Plattform sein, in die Geräte bei Bedarf hinzugefügt oder entfernt werden. Die Geräte werden als Modalität betrachtet.

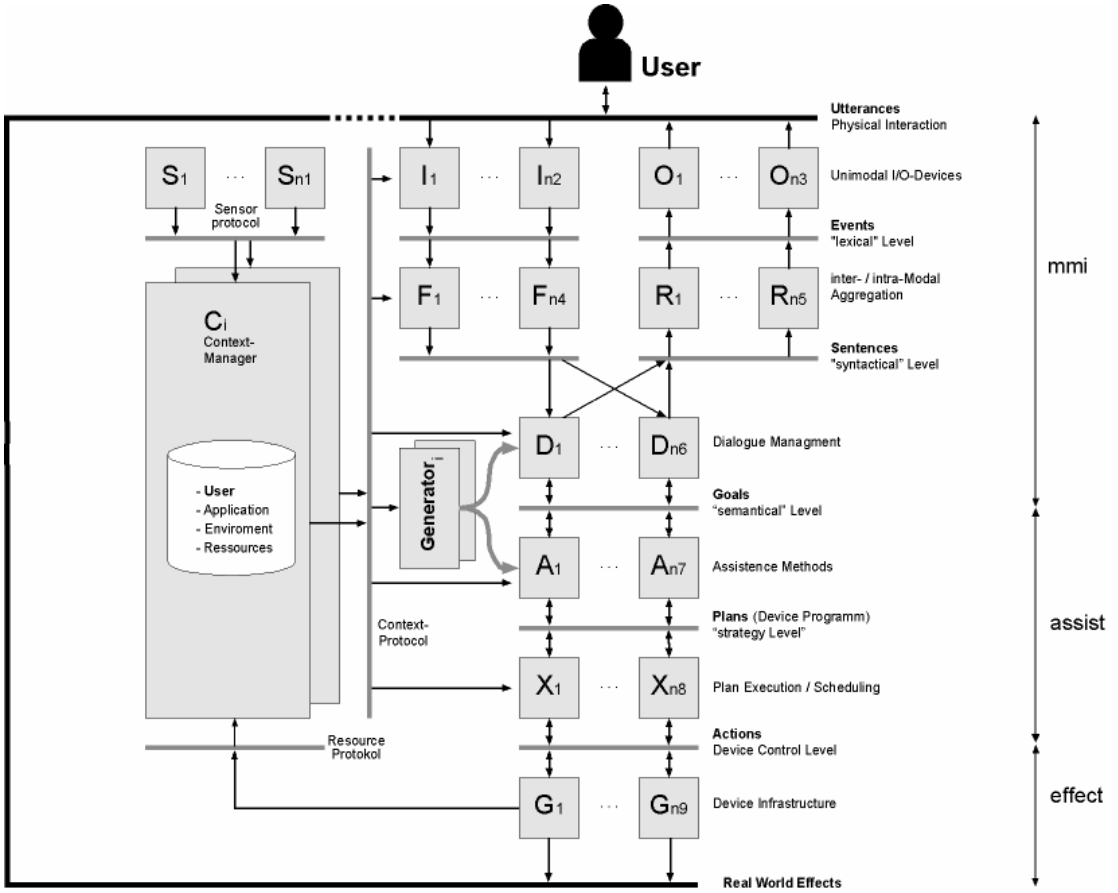

Abbildung 47: Die Architektur des EMBASSI Frameworks [HK02]

4.3.7.3 Aufbau

Das EMBASSI System hat eine übergreifende Sicht auf seine Komponenten und übernimmt die Steuerung und Synchronisation der Modalitäten bzw. Geräte. Entsprechend orientiert sich die EMBASSI Architektur an dem in Kapitel 3.3.6 vorgestellten Pipeline-Modell. Das Architekturbild weist mehrere Ebenen auf, die für Funktionen innerhalb der Pipeline stehen. Die EMBASSI Architektur ist in Abbildung 47 gezeigt.

$I_1..I_{n2}$.stellen die Eingabemodalitäten aller angeschlossenen Geräte dar und erzeugen aus Benutzereingaben so genannte *Events*. $O_1..O_{n3}$ entsprechen den Ausgabemodalitäten. Zusammen bilden sie die Schicht der unimodalen Ein-/Ausgabegeräte.

Die Module $F_1..F_{n4}$ filtern die von $I_1..I_{n2}$ erzeugten Events, etwa um auszuschließen, dass eine Benutzeraktion, die über mehrere Modalitäten aufgenommen wird, vom System mehrfach als Aktion wahrgenommen wird. Sie verschmelzen verschiedene Ereignisse zu einem Gesamtereignis (Data Fusion). Die Ereignisse werden also in eine amodale Form gebracht.

Die Komponenten $D_1..D_{n6}$ bilden die Dialog-Komponente. Sie folgern aus den Ereignissen die Ziele des Benutzers. Die Ziele werden in Sätze dargestellt. Die erwünschte Interpretation aufeinander folgender Ereignisse bedingt ein Gedächtnis in dieser Schicht. $D_1..D_{n6}$ verbinden erstmals die Ausgabeseite mit der Eingabeseite der Interaktion. Wenn Ausgaben erzeugt werden, gelangen diese in der amodalen Form auf die Rendermodule $R_1..R_{n5}$. Die Rendermodule transformieren die Ausgaben in eine für die Komponenten $O_1..O_{n3}$ passende Form. $O_1..O_{n3}$ erzeugen die vom Menschen wahrnehmbaren Ausgaben.

Die im Architekturbild dargestellten Ebenen *A-G* stellen die so genannten Assistenzebenen dar. Die Module der Assistenzebenen sind Agenten, die Ziele realisieren sollen, welche durch die multimodalen Ebenen erkannt werden.

4.3.7.4 Kommunikation

Die Kommunikation zwischen den Systemkomponenten basiert auf der Verwendung von TCP/IP. Das wiederum bildet die Voraussetzung für die Nutzung von KQML für wissensbasierte Module. Als gemeinsame Inhaltssprache wird XML verwendet. Zur Definition der relevanten Ontologien in EMBASSI wird Beschreibungslogik verwendet. Um die Anwendungen der Einzelkomponenten innerhalb des komplexen Systems von EMBASSI dynamisch zu deklarieren, wird ADL (Action Description Language) genutzt.

4.3.7.5 Klassifikation

Die Interaktion im EMBASSI System kann auf ein modellbasiertes UIMS zurückgeführt werden. Für die Generierung von Benutzungsoberflächen werden Task- und Kontextmodelle verwendet. Zur Beschreibung der Interaktionsmöglichkeiten wird KQML eingesetzt.

EMBASSI realisiert Ansätze zur adaptiven Nutzung von mobilen Endgeräten mit Agenten [RE03] und assoziiert die Modalitäten mit den Geräteschnittstellen. Die ad hoc Erweiterung des Systems durch neue Geräte und Komponenten benötigt daher die dynamische Adaption der neuen Geräteschnittstellen.

EMBASSI benutzt stationäre Softwareagenten, die Zugriff auf mobile Endgeräte haben. Da das System im Heimbereich mit stationären Agenten gut arbeiten kann, ohne die zusätzlichen Anforderungen einer mobilen Agentenplattform erfüllen zu müssen, ist die Agentenmigration nicht erforderlich. Aus diesem Grund ist in EMBASSI keine vollständige Migration von Dialogen realisiert.

4.4 Diskussion

Die vorgestellten Beschreibungssprachen und Systeme zur der Erstellung von Benutzungsoberflächen werden nun diskutiert. Es wird untersucht, welche Möglichkeiten vorliegen, welche Punkte hervorzuheben sind und welche Kritik an diese Systeme zu stellen ist.

4.4.1 Beschreibungssprachen

Ein Motiv für die Anwendung einer Beschreibungssprache ist die Reduktion des Entwicklungsaufwands der Benutzungsoberflächen und damit die Minimierung der Kosten. Zu diesem Zweck kommen deklarative Sprachen zum Einsatz, die den Entwicklungsprozess und den Produktzyklus unterstützen können. Diese Ziele werden durch XIML und UIML verfolgt.

XIML soll den kompletten Entwicklungszyklus unterstützen und es ermöglichen, auf die Daten vorhandener Projekte zurückzugreifen, um sie wieder zu verwenden. Dies unterstützt die Migration von Daten aus vorhandenen Projekten in Neuentwicklungen hinein und bewährte Konzepte können weiter genutzt werden. Bei der Portierung von Anwendungen auf neue Zielplattformen ist die Anpassung der Interaktionsdaten ebenfalls gewünscht. Eine automatisierte Anpassung von Interaktionsdaten an neue Umgebungsbedingungen senkt die Kosten und die Dauer einer Portierung.

Ein weiterer Anwendungsbereich ist das Multi-Channeling. Unternehmen können sich durch die Möglichkeit, den Kunden über zusätzlich Informations- und Kommunikationsdienste besser zu erreichen, entscheidende Vorteile verschaffen. Dies bedingt die Unterstützung der Frontendgeräte. Softwareanwendungen sollen sich an die verschiedenen Frontend-Plattformen, wie PC und Web, oder auch an WAP-Mobiltelefone anpassen lassen. Viele Anwendungen müssen mehrere solcher Frontends unterstützen können. Bei der Migration einer Web-basierten Bestellabwicklung vom Desktop-PC zu einem PDA kann der funktionale Kern der Anwendung gleich bleiben. Die Präsentation muss hingegen an die neue Situation angepasst werden. Dies kann in einem modularen System durch den Austausch der Präsentationslogik geschehen. Wenn die Präsentationslogik adaptive Verfahren unterstützt, ist der Austausch nicht notwendig. Die Anwendung kann zudem auf den gleichen funktionalen Daten arbeiten, weshalb die Wartungs- und Aktualisierungskosten minimiert werden. Dieses Ziel wird durch die Sprachen AUIML, UIML und AIML verfolgt.

AAIML stellt bezüglich seines Anwendungsbereichs, nämlich die Unterstützung von behinderten Menschen an öffentlichen Automaten, einen Sonderfall dar. Die Bedienung etwa eines Fahrkartenautomaten kann erleichtert werden, indem der Anwender sein persönliches Endgerät mitbringt. Die Eingabetechnologie kann dadurch individualisiert werden. Das Endgerät kann zudem ein Profil beinhalten, um die Anpassung weiter zu verbessern. Dieser Anwendungsbereich weist ähnliche Anforderungen auf wie das Multi-Channeling.

Unabhängig von den bereits genannten Anwendungsbereichen sind die Browser-basierten Anwendungen einzuordnen. Diese Web-Anwendungen basieren auf einer Client-Server

Architektur, bei denen der funktionale Kern zentral auf dem Server verbleiben kann. Die Programme auf den Endgeräten übernehmen die Aufgaben der Dialogkomponente, der Präsentationskomponente und des Interaction Toolkits. Dies ist das Einsatzgebiet der Sprachen XFORMS und XUL. Die Endgeräte sind auf eine fortwährende Verbindung zum Server angewiesen und somit nicht auf den Offline-Betrieb ausgerichtet. XFORMS dient der Verwirklichung von Web-basierten Formularen, was in Verbindung mit dem Browser im weitesten Sinn als ein Dokumentenmodell mit einem UIMS verstanden werden kann. XUL hingegen ist sehr deutlich auf typische Bedienelemente von WIMP-Systemen ausgelegt und daher eine toolkitorientierte Sprache. Weder XFORMS noch XUL haben einen hohen Abstraktionsgrad.

4.4.2 Systeme

Eine genauere Betrachtung der vorgestellten Systeme ergibt, dass die in Kapitel 2 identifizierten Anforderungen jeweils zum Teil erfüllt sind.

Der Ubiquitous Interactor (UBI) ist ein UIMS auf der Basis eines Dialogmodells und benutzt eine abstrakte Beschreibungssprache für die Benutzungsoberflächen. Die Erzeugung der Benutzungsoberfläche für bestimmte Endgeräte wird jeweils von sog. Interaction Engines vorgenommen. Auch können mehrere Engines parallel geschaltet arbeiten, z.B. um die Modalitäten einer Präsentation gezielt zusammenstellen zu können. Die Engines implementieren keine Mechanismen, um die Interaktion gegenseitig zu koordinieren, wie es von einem multimodalen System zu erwarten ist (vgl. 3.3.7).

ITS ist ein modellbasiertes UIMS auf der Basis verschiedener Modelle. Es erlaubt die Anpassung von Benutzungsoberflächen für Desktop-Systeme und konzentriert sich auf Problemstellungen wie Farbanpassung oder die Skalierung unterschiedlich großen Bildschirmen. Mobile Endgeräte sind bei ITS nicht in Betracht gezogen.

MUSA ist ein UIMS auf der Basis eines Dialogmodells und ermöglicht die Adaption von Benutzungsoberflächen für Web-Applikationen. Das MUSA-System bietet ein flexibles Tailoring-Verfahren an. Das Verfahren nutzt Informationen über die Dialogstruktur für die verbesserte Sequenzialisierung der Dialogelemente. Das Verfahren ist geeignet, um komplexe Dialoge auf kleinen Anzeigen darstellen zu können. MUSA adressiert typische Web-Applikationen, die eine Client-Server Architektur verwenden. Die Migration der Dialoge auf andere Endgeräte bei Erhaltung des Dialogzustands wird nicht unterstützt.

Die Benutzerinteraktion im .NET-Framework erfolgt auf der Toolkit-Ebene. Sog. Windows-Formulare können ihre Benutzungsoberflächen lediglich durch die Skalierung der grafischen Dialogelemente anpassen. Web-Formulare beherrschen die Adaptionsverfahren *Pagination* und *Chunking,* die jedoch wenig flexibel sind, wodurch die Interaktion unkomfortabel werden kann. Wie bei MUSA basieren die Web-Formulare auf einer eine Client-Server Architektur und können daher nicht ohne Netzverbindung bearbeitet werden.

Visual Obliq-und das System von Bharat und Cardelli ist ein Toolkit-basierter Ansatz migrationsfähige Applikationen zu unterstützen. Der niedrige Abstraktionsgrad der Interaktionsdaten schließt jedoch eine Adaption der Benutzungsoberflächen gerade an mobile Endgeräte nahezu aus. Trotzdem ist dieses System interessant, weil es die Idee von kontinuierlichen Dialogen realisiert.

EMBASSI stellt einen modellbasierten Ansatz auf der Basis vieler zusammenwirkender Agenten dar. Die Agenten realisieren ein multimodales UIMS auf der Basis von Benutzer-, Task- und Kontextmodellen. EMBASSI braucht die Adaption, um die der Modalitäten der Endgeräte koordinieren zu können. Das EMBASSI-System basiert auf stationären Agenten und es fehlt der Mobilitätsaspekt der Agenten, der in dieser Arbeit gefordert wird.

SmartKom kann ebenfalls den modellbasierten Systemen zugeordnet werden. Es realisiert einen multimodalen Ansatz und bedingt eine sehr leistungsfähige Infrastruktur. Dies geht darauf zurück, dass insbesondere komplexe sprachlich formulierte Eingaben von System verstanden werden, die eine hohe Leistungsfähigkeit des Endgeräts voraussetzen und derzeit ohne Online-Verbindung nicht zu realisieren ist.

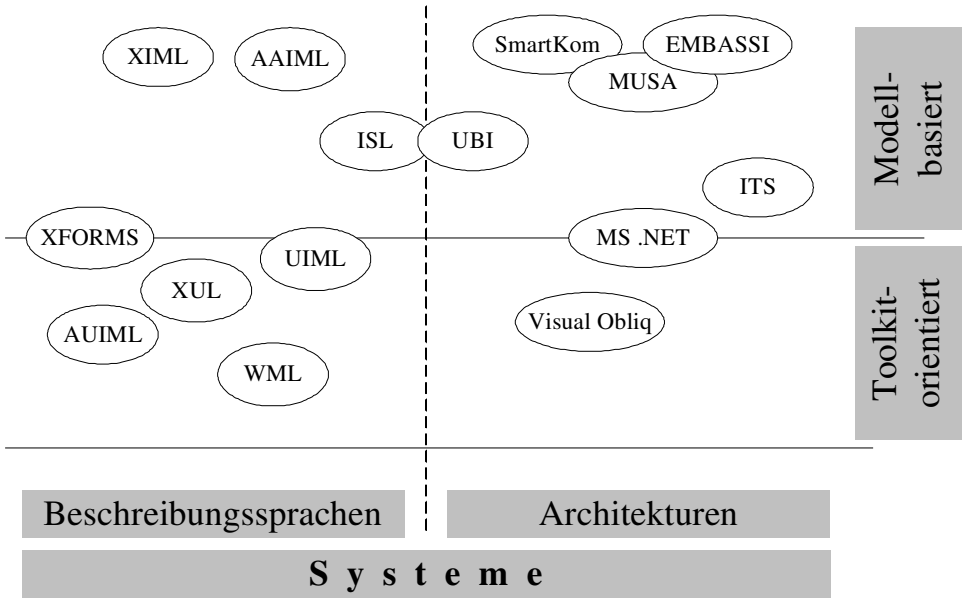

Abbildung 48: Einordnung der Techniken

Das Ordnungsschema aus Abschnitt 4.2.1.3 kann nun genutzt werden, um alle Techniken einzuordnen. Das Schema unterscheidet einerseits zwischen Beschreibungssprachen und Systemen, andererseits nach dem Abstraktionsgrad. Der Abstraktionsgrad orientiert sich daran, ob ein Modell oder ein Toolkit die Basis für die Beschreibung der Benutzungsoberfläche ist. Die Einordnung der Beschreibungssprachen und der Systeme ist in Abbildung 48 vorgenommen.

4.4.3 Fazit

Die vorgestellten Techniken und Systeme erfüllen die Anforderungen für die Migration von Dialogen auf unterschiedliche Endgeräte jeweils nur zum Teil. Entweder werden keine brauchbaren Mechanismen zur Adaption der Endgeräte angeboten, es wird auf die Migration verzichtet, oder es wird der fortwährende Zugriff auf eine leistungsfähige Infrastruktur benötigt.

Aus den Anforderungen und den Ausführungen zum Klassifikationsschema in Abschnitt 4.1.5 geht hervor, dass ein modellbasierter Ansatz gewählt werden sollte, um die Interaktionsdaten möglichst abstrakt halten zu können. Als Ausgangsbasis kann ein Dialog- und Kontextmodell gewählt werden, da es die Aspekte Migration (Kontext) und Interaktion (Dialog) berücksichtigt.

5 Lösungskonzept

Nach der Betrachtung des Stands der Technik in 4 wird nun ein Konzept für ein einen User Interface Agenten (UIA) für mobile und stationäre Endgeräte vorgestellt. Das Kapitel beginnt mit einigen Vorüberlegungen. Danach wird der Begriff des UIA definiert und es erfolgt die schrittweise Vorstellung des Konzepts für die Komponenten des UIA. Wurde der UIA zunächst noch Komponentenweise betrachtet, so finden sich am Ende dieses Kapitels eine Zusammenfassung der Komponenten zu einer Architektur und eine Darstellung als Ganzes. Diese Architektur ist die Grundlage für die in Kapitel 6 vorgenommene Realisierung.

5.1 Vorüberlegungen

Zunächst soll der Stand der bisherigen Überlegungen zusammengefasst werden. Wie in der Einleitung dargelegt und in Kapitel 3 begründet und in Kapitel 4 anhand bestehender Systeme untersucht, ist es sinnvoll, die in Kapitel 3 gezeigten vorhandenen Ansätze, Methoden und Basistechnologien so zusammenzuführen, dass die in Kapitel 2 dargestellten Anforderungen erfüllt werden können. Nun kann die in Kapitel 4 anhand des Stands der Technik identifizierte Lücke zwischen bestehenden Ansätzen und Systemen und der Zielvorgabe dieser Arbeit geschlossen werden.

Die Lösung aller in Kapitel 2 dargestellten Probleme mit einem einzigen System ist nicht zielführend. Das resultiert aus den unterschiedlichen Anforderungen, die an den Ansatz gestellt sind. Eine Verbesserung in einer Hinsicht zieht systematisch eine Verschlechterung in anderer Hinsicht nach sich.

- Die optimale Unterstützung eines Typs von Benutzungsoberflächen schlägt sich in der Anpassung der Beschreibungssprache an spezifische Anforderungen nieder und widerspricht damit der Forderung nach Abstraktion.
- Die Einfachheit der Programmierung von Benutzungsoberflächen steht der Leistungsfähigkeit und Mächtigkeit der Beschreibungssprache entgegen.
- Des Weiteren sind technische Probleme zu nennen, auf absehbare Zeit wird der Wunsch nach vollständiger Unterstützung der natürlichen Kommunikation dem technisch Realisierbaren widersprechen.
- Dazu ist die Dynamik in der Entwicklung der Informationstechnik und insbesondere der Telekommunikation und mobilen Datenverarbeitung zu nennen.

Ein allgemeiner Anspruch auf Berücksichtigung aller heutigen und künftigen Geräte kann im Rahmen dieser Arbeit also nicht erhoben werden. Auch die Spannbreite bei den technologischen Grundlagen und Konzepten (Spannbreite des Agentenbegriffe, unterschiedliche Herangehensweisen bei der Umsetzung von Sprachen zur Beschreibung von Interaktionsdaten, etc.) wirft Probleme auf, die durch einen heutigen Ansatz nicht pauschal befriedigt werden können. Ein System, das alle Anforderungen soweit als möglich befriedigt, wäre schwerfällig und widerspräche schnell den Echtzeitanforderungen, denen die Interaktion zwischen Mensch und Maschine vor allen Dingen gerecht werden muss. Aufgrund seiner Komplexität wäre ein solches System zudem schwer nachvollziehbar für die Autoren von Benutzungsoberflächen.

Nun stellt sich zwangsläufig die Frage, wozu dann ein Konzept für einen User Interface Agent überhaupt gut ist, wenn es offensichtlich nur suboptimale Lösungen beschreiben kann? Ein allgemeines Konzept bietet dem Entwickler eine Spannbreite an Ideen für konkrete Anwendungsfälle und kann auch bei der Spezifikation Hilfestellungen und Vorbilder geben. Ein solches Konzept macht deutlich, an welcher Stelle Schnittstellen vorzusehen sind, es erlaubt die Austauschbarkeit einzelner Teile im Sinne der Komponententheorie. Es können Standards für Datenhaltung, -austausch, Fehlerbehebung und gegenseitige Nutzung von Operationen definiert werden.

5.2 Definition des User Interface Agenten

Der User Interface Agent (UIA) kommuniziert mit dem Benutzer und mit den Softwareagenten, für die der UIA die Interaktion ausführen soll. Die Agenten, welche Dienste des UIA in Anspruch nehmen, werden als *Initiatoren* bezeichnet, weil die Dialoge in jedem Fall von ihnen ausgehen und in keinem Fall vom UIA initiiert sind. Diese Agenten sind damit besser von anderen unterscheidbar.

Entsprechend Abbildung 49 erhalten alle Agenten durch den UIA die Möglichkeit zur Benutzerinteraktion. Der UIA verwaltet die Anfragen der Agenten und führt die Interaktion unter Inanspruchnahme der Interaktionsschnittstellen aus.

Die folgende Definition erfolgt auf Basis der in Kapitel 2 formulierten Anforderungen und bildet die Grundlage für das zu entwickelnde Konzept sowie der dazugehörigen Systemarchitektur:

Definition 3: Ein User Interface Agent (UIA) ermöglicht die Ausführung von *mobilen Dialogen* für Softwareagenten. Mobile Dialoge zeichnen sich dadurch aus, dass sie von einem Gerät auf ein anderes Gerät migriert werden können. Das Zielgerät kann sich bzgl. seiner Interaktionsschnittstellen vom Ausgangsgerät unterscheiden.

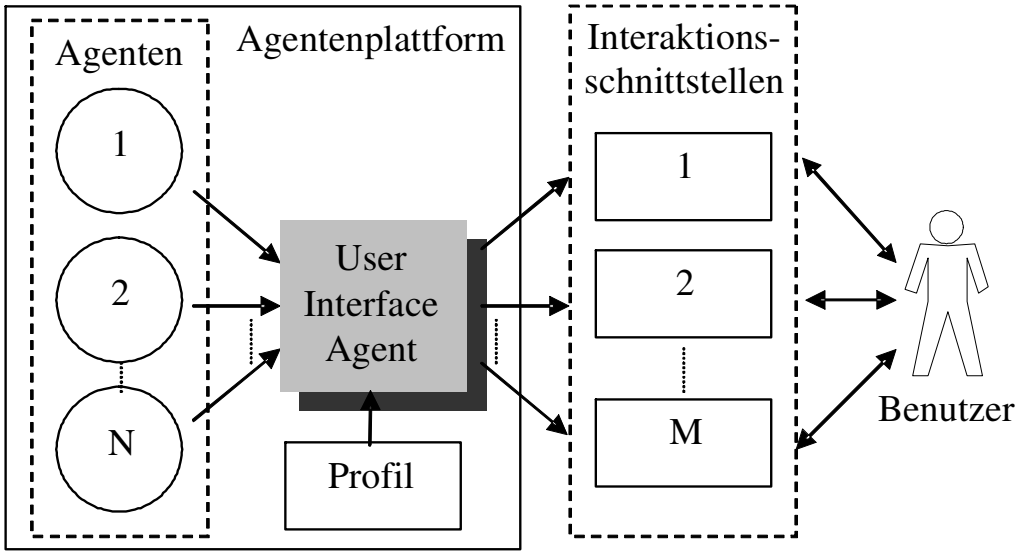

Abbildung 49: Einbettung des mobilen UIA in ein Agentensystem

Gemäß Definition 3 besitzt der geforderte UIA die folgenden Attribute.

- *Migrationsfähigkeit*
  Der UIA ermöglicht Dialoge, die vom Benutzer auf einem Gerät gestartet und auf einem anderen Gerät weitergeführt werden können.
- *Kontinuität*
  Der UIA kann einen Dialog an der Stelle weiterführen, an der er zuvor auf einem anderen Gerät unterbrochen wurde.
- *Adaptivität*
  Der UIA kann einen Dialog an die Interaktionsschnittstellen eines Endgeräts anpassen.

Neben den gerade aufgeführten Eigenschaften entstehen noch weitere Eigenschaften aus den Anforderungen des Kapitels 2.

- *Modularität*
  Der UIA besteht aus wohl von einander abgegrenzten Modulen, die bei Bedarf austauschbar sind.
- *Erweiterbarkeit*
  Der UIA kann nachträglich um neue Module, etwa zur Präsentation, erweitert werden.
- *Autorenumgebung*
  Der UIA verwendet Interaktionsdaten, die mit einfachen Mitteln erstellt und erprobt werden können.

5.3 Dialogmodell

In diesem Abschnitt wird der Dialog als Basis für die Interaktion modelliert. Wie in Abschnitt 3.1.2 dargestellt kann natürlichen Dialogen eine gewisse Struktur zugewiesen werden. Diese Struktur dient als Ausgangspunkt für die Modellierung der Dialoge in der vorliegenden Arbeit.

Der *Dialoganfang* führt nach der Herstellung einer Gesprächsverbindung in die Dialogmitte. Die Herstellung der Verbindung muss von beiden Seiten ratifiziert werden. Für den UIA bedeutet dies die Verifikation der Benutzeridentität und die Annahme des Angebots durch den Benutzer.

Die *Dialogmitte* ist ein Gesprächsabschnitt, der Dialogschritte enthält.

Ein *Dialogschritt* kann die *Eingabe* eines beliebigen Datentyps beinhalten. Außerdem kann ein Dialogschritt seinerseits beliebig viele weitere Dialogschritte enthalten. Durch diese Anordnung weisen Dialoge dieses Modells zunächst eine hierarchische Struktur auf, die sich nicht notwendigerweise in natürlichen Dialogen wieder findet. Um sich einer freien Ordnung der Dialogschritte zu nähern, wird diese Struktur ergänzt. Jeder Dialogschritt erhält die Möglichkeit zu Querverweisen auf beliebige andere Dialogschritte (Schleifen/Verweise→Hyperlinks (vgl. Abschnitt 3.1.2.2)).

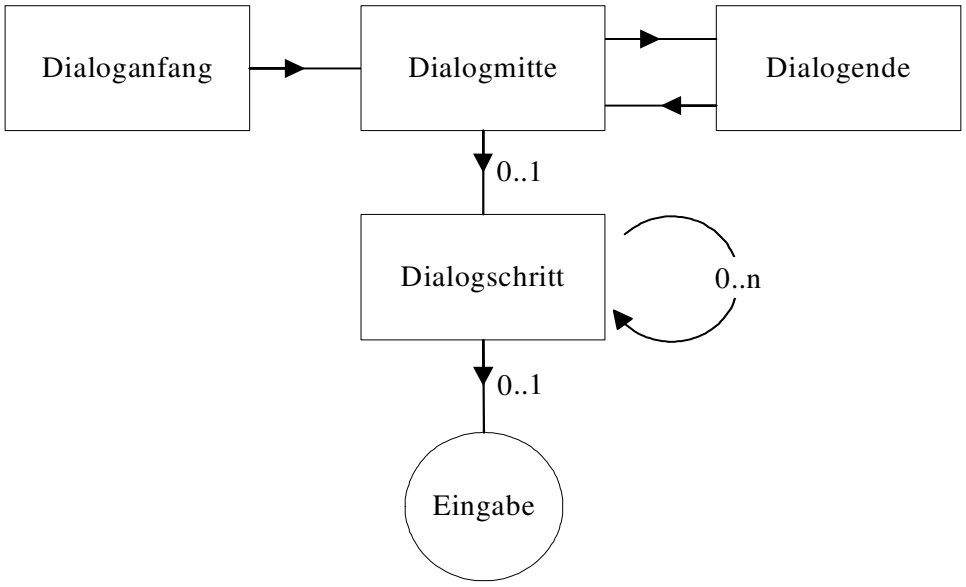

Abbildung 50: Modell eines Dialogs

Das in Abbildung 50 dargestellte Schema zeigt das Modell eines Dialoges.

Beim *Dialogende* kann die Gesprächsverbindung aufgelöst werden. Dies bedarf der gegenseitigen Ratifizierung durch Benutzer und System. Analog zu natürlichen Dialogen kann zu einem beliebigen Dialogschritt des Dialoges zurückgesprungen werden.

Jeder Dialogschritt ist gleichzeitig ein *Dialogziel,* das optional oder obligatorisch bearbeitet werden kann bzw. muss. Der Wechsel von einem Dialogelement in ein Anderes wird als Navigation bezeichnet. Für die Navigation werden gesonderte Funktionen bereitgestellt.

Die Ordnung ist wie in einem natürlichen Dialog nicht auf ein bestimmtes (z. B. hierarchisches) Ordnungsschema eingeschränkt. Deshalb spielt die Navigation eine bedeutende Rolle. Bei Berücksichtigung der Anforderungen an die Navigation ergibt sich für einen Dialogschritt ein Bild entsprechend Abbildung 51.

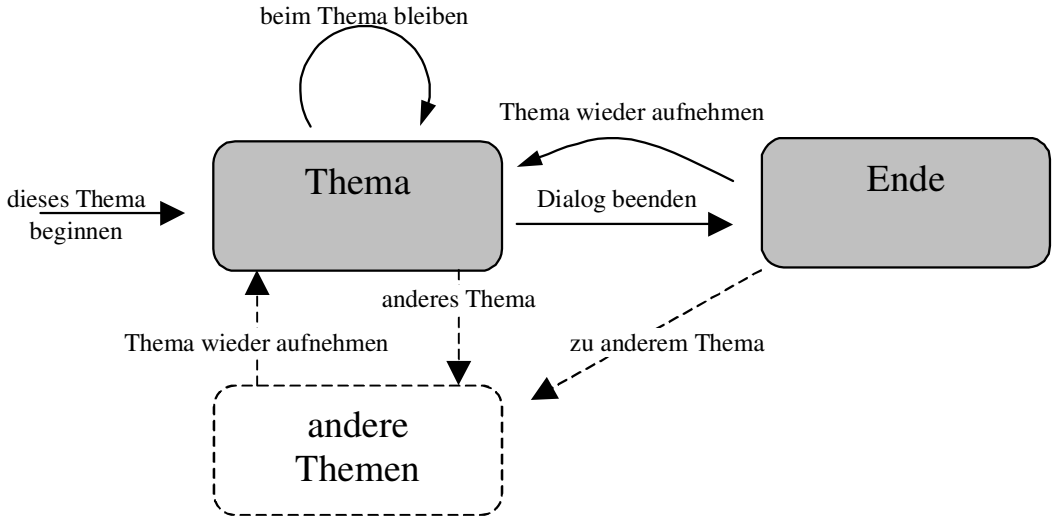

Abbildung 51: Navigationsmöglichkeiten in einem Dialogthema

Die Grammatik der Dialogbeschreibungssprache ist nachfolgend in der erweiterten Backus-Naur-Form (EBNF) nach [ISO14977] wiedergegeben:

```
dialog = „<interface“ struktur eigenschaften „/>“
struktur = „<structure“ dialogstruktur „/>“
eigenschaften = „<style“ attribute „/>“
dialogstruktur = dialoganfang dialogmitte dialogende
dialoganfang = „<part classname=DialogOpening name=“ bezeichner „/>“
dialogmitte =  dialogschritt
dialogschritt = bezeichner [ [ dialogschritt ] | [ eingabe ] ]
dialogende = „<part classname=DialogClosing name=“ bezeichner „/>“
```

```
attribute = eigenschaft | attribute

eigenschaft = „<property partname=“ bezeichner „attribute=“
            attributname „value=“ wert „/>“

eingabe = boolinput | charinput | integerinput | floatinput |
            stringinput | stringlistinput

boolinput = “<part classname=BoolInput name=” bezeichner “/>”

charinput = “<part classname=CharInput name=” bezeichner “/>”

integerinput = “<part classname=IntegerInput name=” bezeichner “/>”

floatinput = “<part classname=FloatInput name=” bezeichner “/>”

stringinput = “<part classname=StringInput name=” bezeichner “/>”

stringlistinput =  „<part classname=StringListInput name=“ bezeichner
                   „/>“

bezeichner = buchstabe | { buchstabe | ziffer }
```

5.4 Einordnung in das ARCH-Modell

Das ARCH-Modell identifiziert und benennt die Komponenten entlang der Kette zwischen Benutzer und Anwendung. Die Komponenten aus Abbildung 49 können wie folgt in das ARCH-Modell eingeordnet werden:

- Der *funktionale Kern* entspricht den Agenten (Initiatoren).
- Die *Adapter Komponente* entspricht dem Message Transport System (MTS) der Agentenplattform.
- Die *Domänen Objekte* werden mit Hilfe der Agentenkommunikationssprache (ACL) ausgetauscht.
- Die *Dialogkomponente* ist ein Teil des UIA.
- Die *Präsentationskomponente* ist wie die Dialogkomponente ein Teil des UIA.
- Das *Interaction Toolkit* wird vom UIA gesteuert. Der UIA kann ggf. mehrere Toolkits bedienen.

5.5 Adaption

Ein UIA soll an neue Endgeräte anpassbar sein. Dies beeinflusst das Konzept erheblich. Die Beschaffenheit von Interaktionsschnittstellen gegenwärtiger und künftiger Endgeräte kann mit einem monolithischen System nicht gewährleistet werden. Um im Vorfeld neue, heute noch

unbekannte Interaktionsmethoden nicht auszuschließen, sollte das nachträgliche *Erweitern des Systems* um neue Präsentationstechniken ein leicht durchzuführender Anwendungsfall sein.

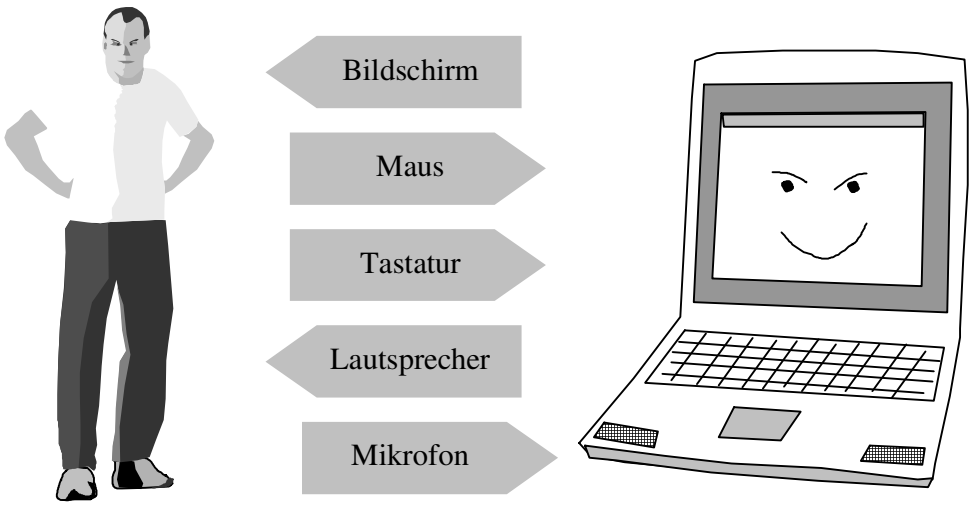

Abbildung 52: Ein Endgerät kombiniert mehrere Modalitäten für die Benutzungsoberfläche

Elektronische Endgeräte kombinieren meist verschiedene Basistechniken für die Interaktion. Mit diesen Basistechniken sind etwa eine PC-Tastatur, ein Maus, Rasterbildschirm, eine Stifteingabe, Handschrifterkenner, etc. gemeint. Entsprechend der Darstellung in Abbildung bedient sich die Interaktion i.a. mehrerer solcher „Modalitäten". Durch die gezielte Unterstützung dieser Modalitäten durch einzelne Module kann eine Vielzahl von Geräten mit einem relativ kleinen Satz an Modulen generisch unterstützt werden.

Wird dieser multimodale Ansatz verfolgt, dann entstehen dem System eine Reihe zusätzlicher Anforderungen:

- Auf unterschiedlichen Modalitäten eingegebene Daten müssen zusammengeführt werden. Ein solcher Prozess wird auch als *Data Fusion* bezeichnet.
- Daten, die ausgegeben werden, müssen hingegen aufgeteilt werden. Dieser Prozess wird als *Data Splitting* bezeichnet.
- Die Bearbeitung mehrerer Modalitäten zur gleichen Zeit bedarf der *Synchronisation der Modalitäten.*

Die Präsentationstechniken können durch entsprechende Module unterstützt werden, die im Folgenden als *Rendermodule*[16] bezeichnet sind. Ein Rendermodul greift auf die Funktionen

[16] Im Gegensatz zur häufig verwendeten Bedeutung des engl. Wortes *Rendern*, die sich lediglich auf das Sichtbarmachen von Daten bezieht, kann ein Rendermodul, Daten auch nicht sichtbar ausgeben und Daten von Benutzer auch entgegennehmen. Bsp.: Sprachrendermodul = Modul das Sprachausgabe unterstützt

eines Interaction Toolkits zurück und adaptiert eine oder mehrere Modalitäten in Kombination. Die Funktionen der Rendermodule werden eine einheitliche Schnittstelle abgebildet.

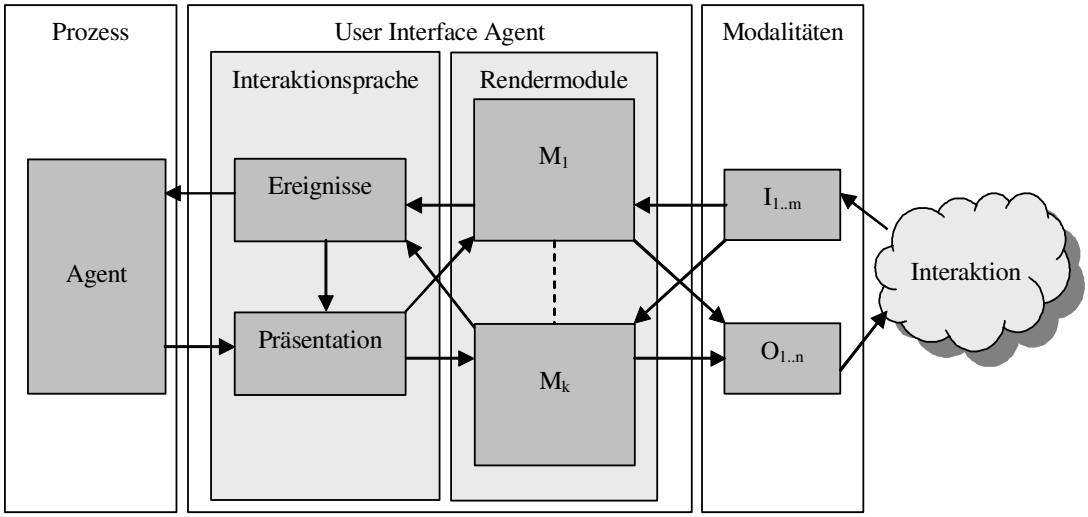

Abbildung 53: Anordnung des mobilen UIMS als Pipeline Modell

Trägt man die Komponenten des UIA in ein Pipeline Modell ein, so ergibt sich eine Anordnung entsprechend Abbildung 53. Ein Initiator fordert den UIA zur Präsentation seiner Interaktionsdaten auf. Zur Beschreibung der Präsentation werden die Interaktionsdaten in einer Beschreibungssprache kodiert. Die vom Benutzer wahrnehmbare Präsentation wird von den Modulen $M_{1..k}$ vorgenommen, die sich der Eingabemodalitäten $I_{1..m}$ und der Ausgabemodalitäten $O_{1..n}$ bedienen. Die Rendermodule verknüpfen die Ein- und Ausgaben miteinander.

Die Rendermodule adaptieren ein bestimmtes Interaction Toolkit. Um verschiedene Rendermodule mit dem System verbinden zu können wird eine Adapterkomponente benötigt. Die Adapterkomponente erlaubt den simultanen Betrieb mehrerer Rendermodule und ermöglichen dynamischen Austausch einzelner Module zur Laufzeit. Die Komposition einer Benutzungsoberfläche aus verschiedenen Rendermodulen ist so möglich. Um die Rendermodule koordinieren zu können, ist eine Synchronisation erforderlich. Auf der Ausgabeseite müssen Interaktionsdaten auf die Modalitäten verteilt werden, auf der Eingabeseite müssen die Daten kombiniert werden.

5.6 Synchronisation der Dialogelemente

Nach Abschnitt 5.3 wird jeder Dialog des mobilen UIMS in Dialogelemente untergliedert. Im vorangegangenen Abschnitt 5.5 wurde bereits die Synchronisationsproblematik angesprochen.

Dieser Abschnitt nimmt sich dieser Problematik an und stellt ein Konzept für die Synchronisation der Dialoge und ihrer Elemente vor.

Die Synchronisation eines Dialogs betrifft alle Dialogelemente, sowohl auf der abstrakten, als auch auf der konkreten Ebene. Dazu ist es zweckmäßig, jeden Dialog einmal abstrakt und einmal konkret für jede Präsentationsform als Objektmodell zu repräsentieren. Dies ist in folgender Abbildung 54 für den Fall von zwei angeschlossenen Rendermodulen skizziert.

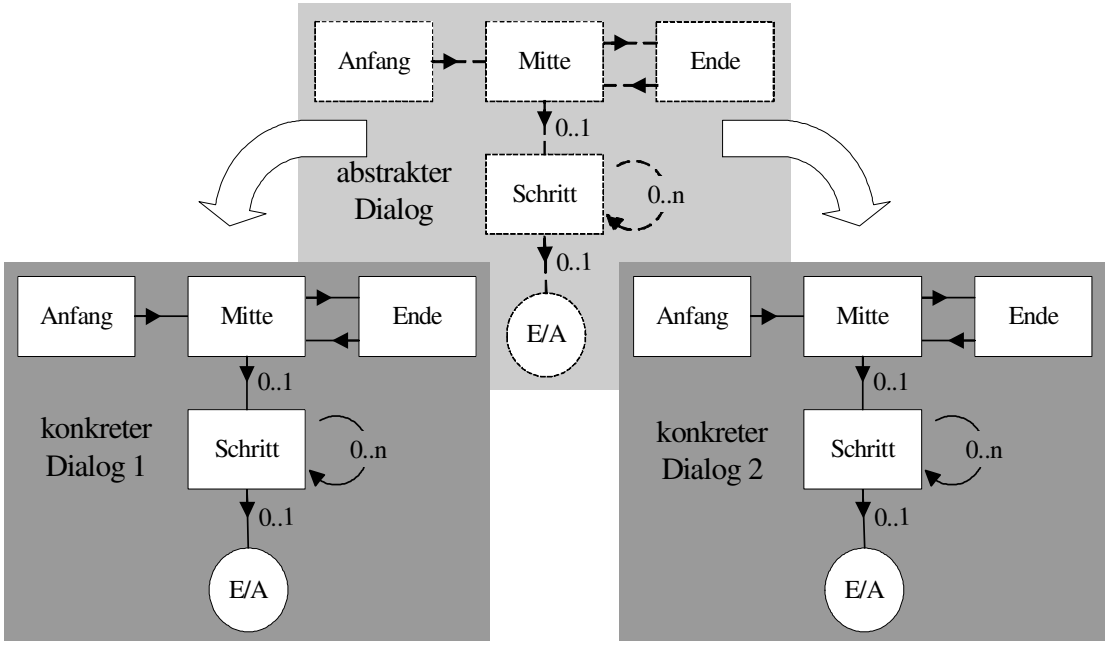

Abbildung 54: Aus einem abstrakten Dialog werden konkrete Dialoge erzeugt

Nachdem für jedes Rendermodul ein konkretes Objektmodell erstellt wurde, werden die Dialogobjekte des abstrakten Dialogs mit den Dialogobjekten der konkreten Präsentation verbunden, so dass jedes abstrakte Element mit seinen konkreten Abbildungen kommunizieren kann. Diese Verbindung ist in Abbildung 55 dargestellt.

Das abstrakte Dialogelement benötigt eine standardisierte Schnittstelle, die zu den konkreten Präsentationselementen führt. Wenn die Verbindung zwischen abstraktem und konkretem Element während der Laufzeit hergestellt und wieder getrennt werden kann, dann ist eine dynamische Adaption realisierbar. Die Ausgabe eines Dialogobjekts sind Ereignismeldungen, die auf Benutzereingaben zurückgeführt werden.

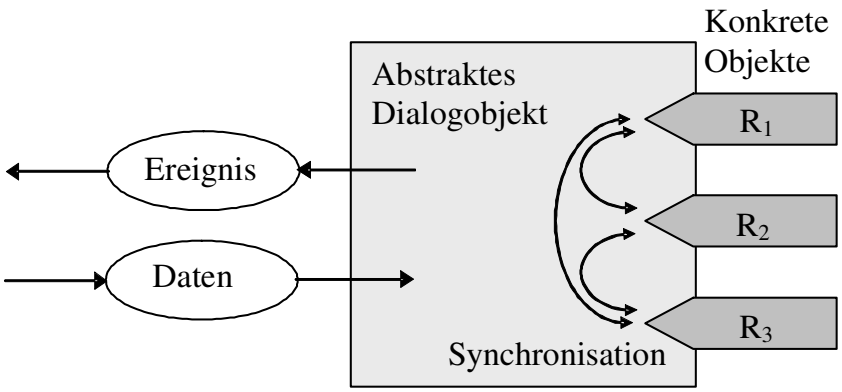

Abbildung 55: Ein abstraktes Element steuert seine konkreten Ausprägungen R1, R2 und R3

5.6.1 Adapterkomponente

Die Aufgabe der Adapterkomponente besteht darin, einen Mechanismus zur Adaption der verschiedenen Rendermodule bereit zu stellen. Die Adapterkomponente wird durch die konsequente Umsetzung des Abstract Factory Patterns (siehe Abbildung 56) bei der Schnittstellendefinition erreicht.

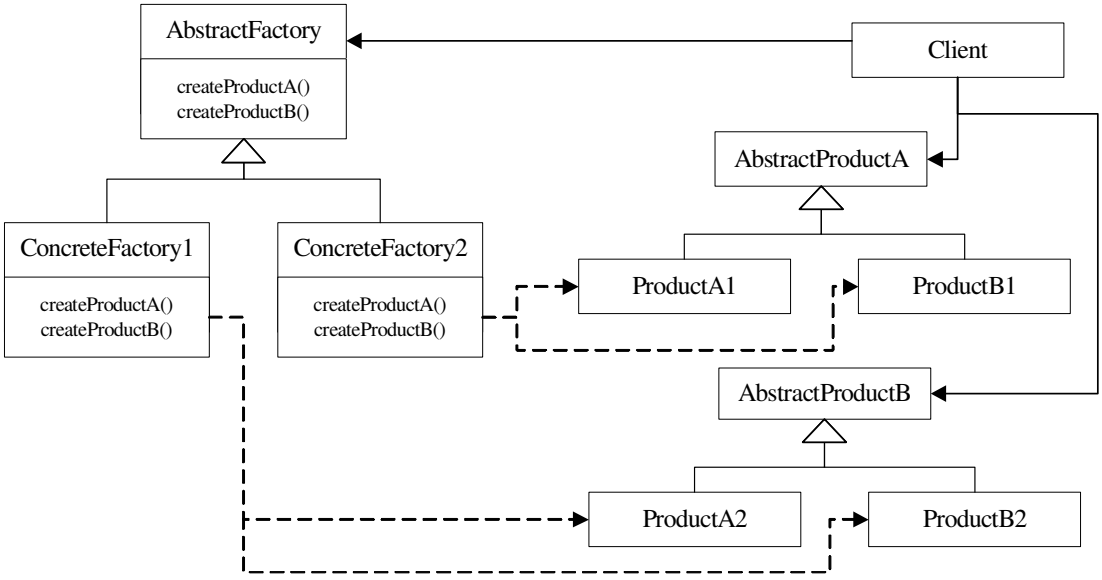

Abbildung 56: Das Abstract Factory Pattern nach [GHJV98]

Das Abstract Factory Pattern stammt aus dem Bereich des Pattern Oriented Software-Engineering: Ein Klient greift auf bestimmte Produkt-Klassen über Referenzen auf entsprechende abstrakte Oberklassen zu. Da die konkrete Produktpalette erst zur Laufzeit bekannt sein kann und auch während der Ausführung jederzeit austauschbar bleiben soll, werden die Produktinstanzen nicht durch direkten Aufruf des Konstruktors gebildet, sondern durch die Benutzung spezieller Factories, die über das nötige Wissen zur Instantiierung der Produktpalette verfügen. Jede Produktpalette besitzt eine eigene Factory. Alle Factories implementieren die *Abstract Factory*, von der der Name des Patterns stammt. Der Klient wendet sich zwecks Instantiierung der Produkte immer nur an die Abstract Factory Klasse. Dadurch kann ein Wechsel der Produktpalette zur Laufzeit einfach durch einen Austausch der sich hinter der Abstract Factory Referenz verbergenden konkreten Factory durchgeführt werden [GHJV98][Busc96].

Das Pattern kann immer dann eingesetzt werden, wenn

1. ein System unabhängig davon sein soll, wie seine Objekte erzeugt, zusammengesetzt oder repräsentiert werden,
2. ein System mit einem bestimmten Satz von Klassen konfiguriert werden können soll.
3. in einem Framework ein Satz von Klassen eingesetzt werden soll, jedoch nur Interfaces davon deklariert werden.

Besonders Punkt 1 und 2 treffen auf den vorliegenden Anwendungsfall in besonderem Maße zu, da abstrakte Dialogelementen zu konkret darstellbaren Dialogelementen transformiert werden sollen.

- *AbstractFactory*
  deklariert ein Interface für die Erzeugermethoden (Factory Methods), welche die Instanzen der Produkte erzeugen.
- *ConcreteFactory*
  implementiert die Factory Methoden zur Erzeugung der konkreten Produkte. *ConcreteFactory1* erzeugt die Produkte *ConcreteProductA1* und *ConcreteProductB1*, *ConcreteFactory2* erzeugt die Produkte *ConcreteProductA2* und *ConcreteProductB2*.
- *AbstractProduct*
  deklariert ein Interface für die Produkte.
- *ConcreteProduct*
  implementiert das Produkt zu einer der abstrakten Klassen *AbstractProduct1* bzw. *AbstractProduct2*.
- Das Client-Objekt benutzt als Typisierung nur die abstrakten Klassen *AbstractFactory* und *AbstractProduct1* und *AbstractProduct2*.

Folgende Vorteile werden durch die Anwendung des Patterns erzielt:

- *Konkrete Klassen werden isoliert.*
  Der Client Code wird unabhängig von den konkreten Produkt-Klassen. Er deklariert nur den Interfacenamen oder die abstrakte Produkt-Klasse, jedoch keine konkreten Produkt-Klassen.
- *Einfaches Auswechseln von Klassen.*
  Der Klassenname der konkreten Produkte erscheint genau einmal im Code, nämlich in der ConcreteFactory. Dadurch wird der Austausch durch eine andere Klasse (beispielsweise eine andere Version des Produkts) massiv vereinfacht. Zudem können alle Produkte gemeinsam ersetzt werden, indem eine andere ConcreteFactory Instanz benutzt wird.

Die Dialogelemente stellen im vorliegenden Fall die Produkte der Rendermodule dar, die als Factory implementiert werden. Zur Entwicklung passender Rendermodule reicht die Einhaltung des Patterns aus.

Die Adapterkomponente verbindet den User Interface Agent mit den Rendermodulen. Die Rendermodule müssen für die Adapterkomponente die folgende Funktionalität bereitstellen:

- Erzeugen eines Dialogs
- Erzeugen eines Dialogschritts
- Erzeugen eines Dialogelements für die Eingabe eines Datentyps
- Erzeugen eines Dialoganfangs und eines Dialogendes
- Bereitstellen von Information über Darstellungsfähigkeiten des Rendermoduls

Die durch die Rendermodule erzeugten Elemente müssen einer Schnittstellenspezifikation folgen, die sich aus den folgenden Abschnitten definiert.

5.6.2 Interaktionsereignisse

Durch die Interaktion werden verschiedenartige Ereignisse ausgelöst. Diese Ereignisse auf der Seite des Benutzers oder auf der Seite des Agentensystems. Die Ereignisse beziehen sich auf die Eingabe von Daten oder auf die Dialogkontrolle und Navigation. Folgende Ereignisse können eintreten:

- Der Dialog wird initiiert.
- Der Dialog wird beendet oder abgebrochen.
- Der Benutzer hat eine gültige Eingabe gemacht.
- Der Initiator hat die Aktualisierung eines Dialogelements angefordert.
- Der Benutzer hat ein Element ausgewählt (Fokuswechsel).
- Die Verbindung zum Benutzer ist unterbrochen.

Diese Ereignisse werden den Dialogelementen signalisiert, damit sie sich auf die Situation synchronisieren können. Dazu werden die Dialogelemente mit entsprechenden Methoden ausgestattet.

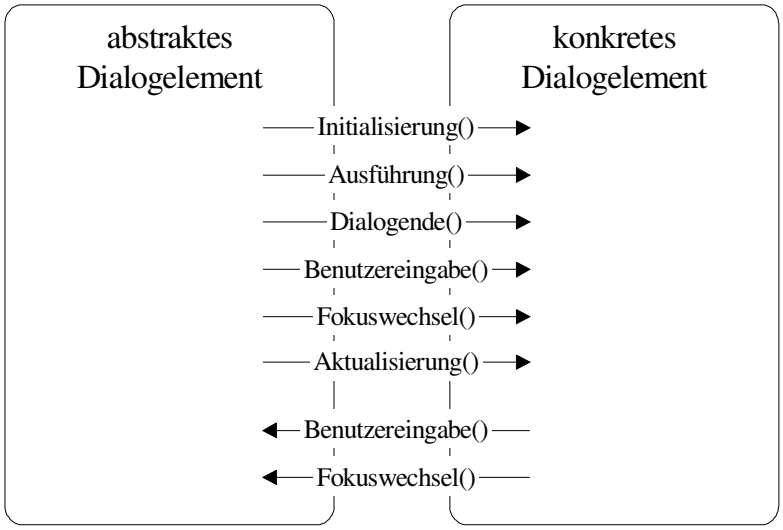

Abbildung 57: Austausch der Ereignisse zwischen den Dialogelementen nach [Blec04]

Die Grundidee hierbei besteht darin, dass das abstrakte Dialogelement die Ereignisse seiner konkreten Repräsentanten sammelt und sie gegenseitig über die Ereignisse, welche durch Benutzeraktivitäten mit anderen Rendermodulen bzw. durch das System ausgelöst wurden, informiert.

Aufgrund der zentralen Position sammelt ein abstraktes Dialogelement die Ereignisse seiner konkreten Repräsentanten. Das abstrakte Dialogelement steuert den Prozessablauf der konkreten Elemente in den Rendermodulen, ermittelt den aktuellen Status und koordiniert die Navigation. Das abstrakte Dialogelement ist somit für die Vorgänge *Data-Splitting* und *Data-Fusion* verantwortlich.

5.6.3 Data Splitting

Die in Abschnitt 5.6.2 dargestellte Schnittstelle wird genutzt, um die Interaktionsdaten, welche in Richtung des Benutzers fließen, aufzuteilen. Jedes abstrakte Element verteilt die Daten an seine konkreten Repräsentanten. Folgende Fälle können hierbei auftreten:

- *Erzeugung eines Dialogelements*
  Bei der Erzeugung eines Dialogelements wird zunächst das abstrakte Dialogelement erzeugt. Beim Aufruf des Konstruktors werden alle Attribute an das Dialogelement übergeben.

Anhand der Liste der registrierten Rendermodule erzeugt das abstrakte Dialogelement nun seine konkreten Repräsentanten und gibt die Initialisierungsinformation weiter.

- *Aktualisierung eines Dialogelements*
  Bei der Aktualisierung eines Dialogelements durch den Initiator, erhält das abstrakte Dialogelement als Übergabeparameter den neuen Wert des Datums. Als Folge ruft das abstrakte Dialogelement die Aktualisierungsfunktion seiner konkreten Repräsentanten auf unter Weitergabe des neuen Datenwerts.
- *Benutzereingabe an einem Dialogelement*
  Wenn ein konkretes Dialogelement eine Benutzereingabe erkennt teilt es dies dem abstrakten Dialogelement mit. Das abstrakte Dialogelement ruft nun die Benutzereingabe-Funktion der anderen konkreten Dialogelemente auf. Hierbei wird als Aufrufparameter das eingegebene Datum übergeben.

5.6.4 Data Fusion

Die in Abschnitt 5.6.2 dargestellte Schnittstelle wird auch genutzt, um die Interaktionsdaten, die aus Richtung des Benutzers zum System fließen, zusammenzufassen. Hierbei kombiniert jedes abstrakte Element die Daten seiner konkreten Repräsentanten.

Um die Eingabe, welche über mehrere Kanäle erfolgen kann, zu koordinieren, wird die zeitliche abfolge der Eingabe berücksichtigt. I.A. ist der zuletzt eingegebene Datenwert, der gültige. Wird auf zwei unterschiedlichen Rendermodulen die Eingabe des gleichen Wertes erkannt, wird die doppelte Aktualisierung des Datums des Dialogelements durch das abstrakte Dialogelement unterdrückt.

5.6.5 Gerätefähigkeiten

Der UIA muss die Möglichkeiten der Endgeräte zur Ein- und Ausgabe bestimmter Datentypen berücksichtigen. Kann ein benötigter Datentyp nicht auf ein bestimmtes Gerät abgebildet werden, muss das System darauf reagieren können. Um den UIA in die Lage zu versetzen die Endgerätefähigkeiten zu erkennen, werden Zusatzinformationen in die Rendermodule eingebracht. Die Zusatzinformation beschreibt die Gerätefähigkeiten nach den darstellbaren Datentypen in Bezug auf Aus- und Eingabe. Sollte ein benötigter Datentyp nicht darstellbar sein, können möglicherweise nicht alle Dialogschritte bearbeitet werden. Dann kann ein Dialog entweder gar nicht oder nur teilweise dargestellt werden.

Ein nur teilweise darstellbarer Dialog könnte aber Dialogteile enthalten, die für sich genommen sinnvoll ausführbar sind. Ein Beispiel hierfür wäre etwa die Darstellung von Nachrichten eines Email-Agenten. Wenn ein solcher Agent in einem Kraftfahrzeug arbeitet, könnte durch den Kontext die Interaktion auf Sprache und wenige Tasten reduziert sein. Die Eingabe einer Email wäre während der Fahrt also nicht möglich. Die Wiedergabe einzelner Botschaften veranlasst

durch den Email Agenten wäre aber eine nützliche und realisierbare Funktion. In diesem Falle würde das Rendermodul, welches die Interaktion im Fahrzeug realisiert aufgrund des Kontexts die Eingabe eines Textes und damit das Erstellen einer Nachricht nicht erlauben. Das Wiedergeben einer Email und das Umherschalten zwischen verschiedenen Nachrichten wären dagegen erlaubt.

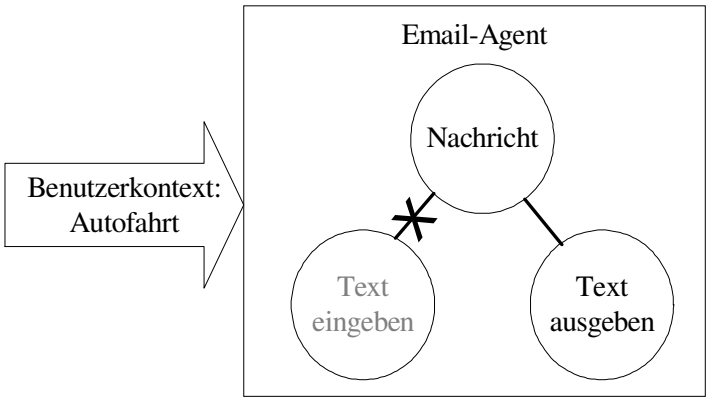

Abbildung 58: Der Zugang zu Dialogteilen kann kontextabhängig sein

5.7 Adaptionsverfahren

Die Interaktionsmechanismen mobiler Endgeräte sind insbesondere auf begrenzte Anzeigeflächen zugeschnitten. So verfügen heute übliche Desktop-PC über eine Darstellungsauflösung von 1280x1024 Bildpunkten[17], während ein handelsüblicher PDA mit etwa 320x320 Bildpunkten auskommen muss[18], was eine Reduktion etwa um den Faktor 12 bedeutet. Die Darstellungsfähigkeiten von Mobiltelefonen sind dem gegenüber noch einmal stark reduziert und markieren mit Auflösungen von etwa 130x130 Bildpunkten[19] die untere Grenze der eingesetzten Displays.

Es ist offensichtlich, das die begrenzte Auflösung der Anzeige zu Problemen, insbesondere bei sehr umfangreichen Benutzungsoberflächen, führt. Um dieses Problem zu lösen kommt das in [BS03] beschriebene Verfahren zur Anwendung.

---

[17] Dies entspricht der Auflösung eines heute üblichen TFT-Displays mit einer Bildschirmdiagonale von 17 Zoll

[18] Beispiel: Der PDA „Tungsten C“ der Fa. Palm kann 320x320 Bildpunkte bei 65535 darstellbaren Farben auf einer Fläche von etwa 3 Zoll

[19] Beispiel: Das Siemens SL65 stellt bei einer Auflösung von 130x130 bei 65536 Farben 7 Textzeilen dar

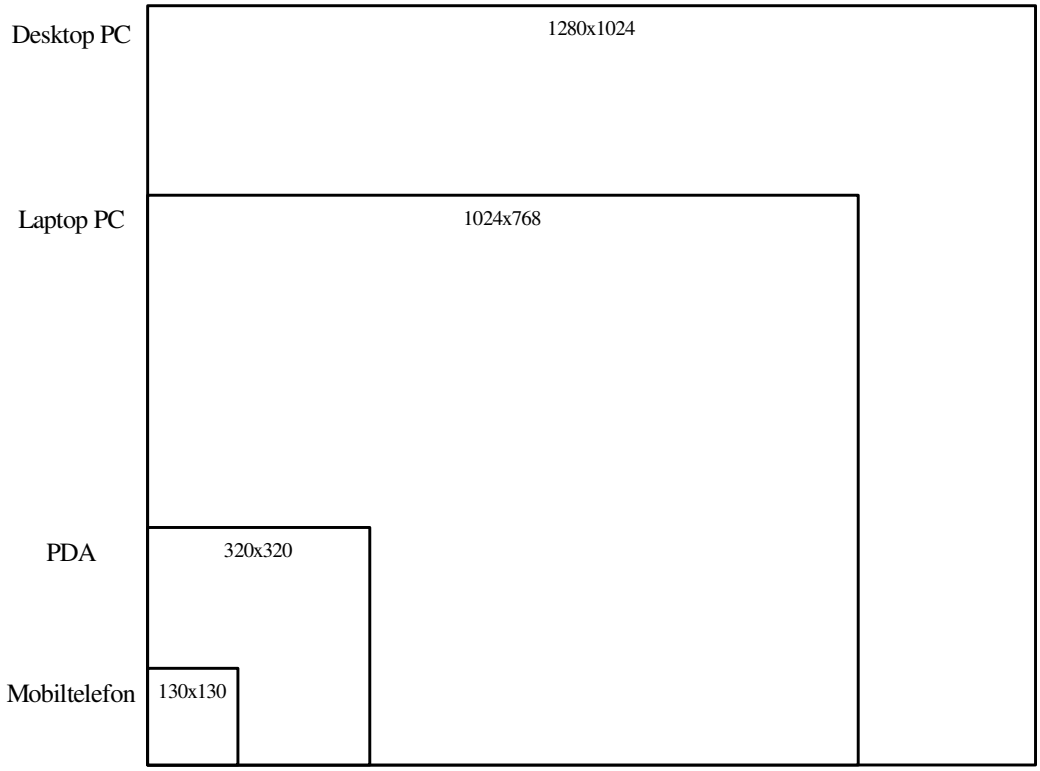

Abbildung 59: Proportionaler Vergleich der heute üblichen Anzeigeauflösungen

Diese Methode minimiert die Anzahl der gleichzeitig darzustellenden Dialogelemente. Die Reduktion der Dialogelemente kann aufgrund des Interaktionsverlaufs und der Dialogstruktur vorgenommen werden. Dabei wird Bezug zum Mechanismus der Verweise in natürlichen Dialogen in Abschnitt 3.1.2 genommen.

Um die im Rahmen dieser Arbeit verwendete Dialogstruktur zu beschreiben wird im Folgenden auf die Terminologie der Graphentheorie zurückgegriffen (vgl. [Dist00]).

Einen Dialog wird als Graph $G = (V, E)$ betrachtet.

Die Dialogelemente entsprechen hierbei den Knoten *V* und die Beziehungen, die zwischen den Dialogelementen bestehen sind seine *Kanten E*.

Werden die zusätzlich in die Dialoge eingebrachten Verweise ausgegrenzt, weißt die Dialogmitte eine hierarchische Struktur auf und entspricht daher einem *Baum*.

Die Dialogschritte, die der Eingabe dienen sind *Blätter*, da sie nur *eine* Kante besitzen und daher den Grad 1 aufweisen.

Die Dialogstruktur soll die inhaltlichen Bezüge der Dialogschritte abbilden. Ein Beispiel hierfür ist ein Dialog für ein Kinokarten-Reservierungssystem. Ein Film ist assoziiert mit Termin(en) für die Vorführung. Eine Kinokarte ist assoziiert mit einem Termin, mit einem Saal für die

Vorführung und mit einem Sitzplatz. Eine Reservierung assoziiert eine Person mit einer Kinokarte. Ein Saal ist assoziiert mit Filmen, usw. Das Beispiel des Kinokarten-Reservierungs Ubiqitous systems kann auf einen Baumgraphen abgebildet werden.

Werden nun zusätzliche Verweise innerhalb des Dialogs berücksichtigt, können *Kreise*[20] entstehen und der Dialoggraph ist kein Baum mehr, wie in Abbildung 60 gezeigt. Dies bringt mehrere Probleme mit sich. Der Dialog konnte ohne die Verweise auf einem Gerät mit entsprechender Anzeigenauflösung Präsentiert werden. Sind die Verweise berücksichtigt, können Kreise entstehen. Weil die Darstellung des Dialogs auf einer *Traversierung* des Dialoggraphen beruht, sind solche Kreise nicht ohne weiteres darstellbar.

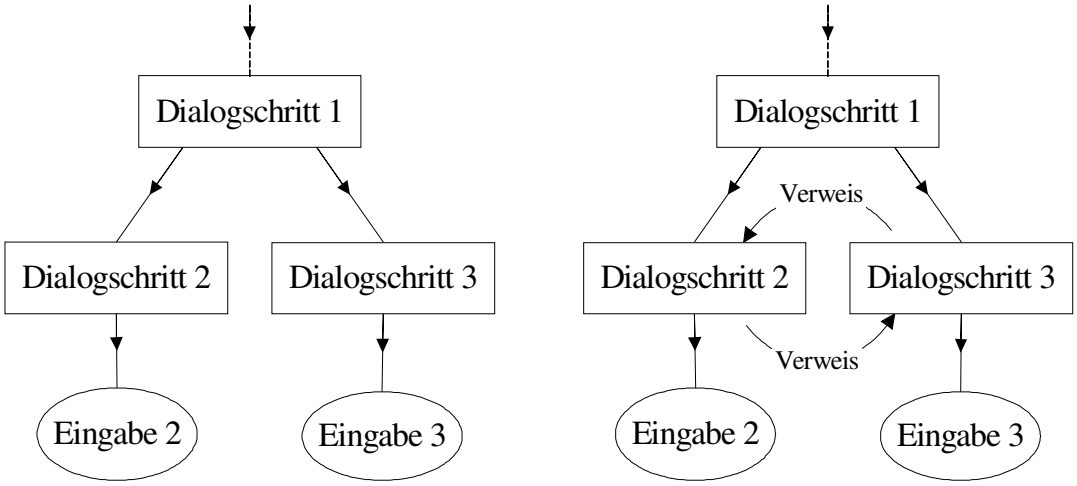

Abbildung 60: Dialogmitte ohne und mit Kreis

Um dieses Problem und das Problem der Reduktion der anzuzeigenden Dialogelemente zu lösen, werden die Begriffe *Point of View (PoV)* und *Range of View (RoV)* eingeführt. *PoV* bezeichnet den aktuell vom Benutzer fokussierten Dialogschritt. *RoV* gibt die Weglänge an, die vom aktuellen Dialogschritt ausgehend visualisiert werden soll. Ist *RoV=1*, dann können keine Kreise mehr entstehen. In diesem Fall ist die Anzahl der gleichzeitig darzustellenden Elemente ebenfalls minimal. Bei *RoV=1* stellt diese Methode ein Verfahren dar, die Interaktion in eine Sequenz umzusetzen. Diese Methode kann gleichermaßen auf grafische und sprachliche Interaktionsformen angewendet werden.

Die Implementation des Verfahrens erfolgt innerhalb eines Rendermoduls. Das Fokussierte Dialogobjekt fordert seine (innerhalb des ROV sichtbaren) benachbarten Dialogobjekte zur Präsentation auf (siehe dazu Abbildung 61).

[20] Diese Kreise werden oft auch als Zyklen bezeichnet

Für die Funktion des Verfahrens ist die Kenntnis des aktuell ausgeführten Dialogobjekts wichtig. Dies kann über den Fokuswechsel[21] ermittelt werden. Ein Fokuswechsel wird dem Rendermodul entweder durch den Aufruf der Fokuswechsel() bzw. Interpret()-Methode des betroffenen Dialogobjekts (vgl. Abschnitt 5.6.2) mitgeteilt, oder das Rendermodul hat den *Fokuswechsel* selbst aufgenommen und verfügt daher bereits über die Information.

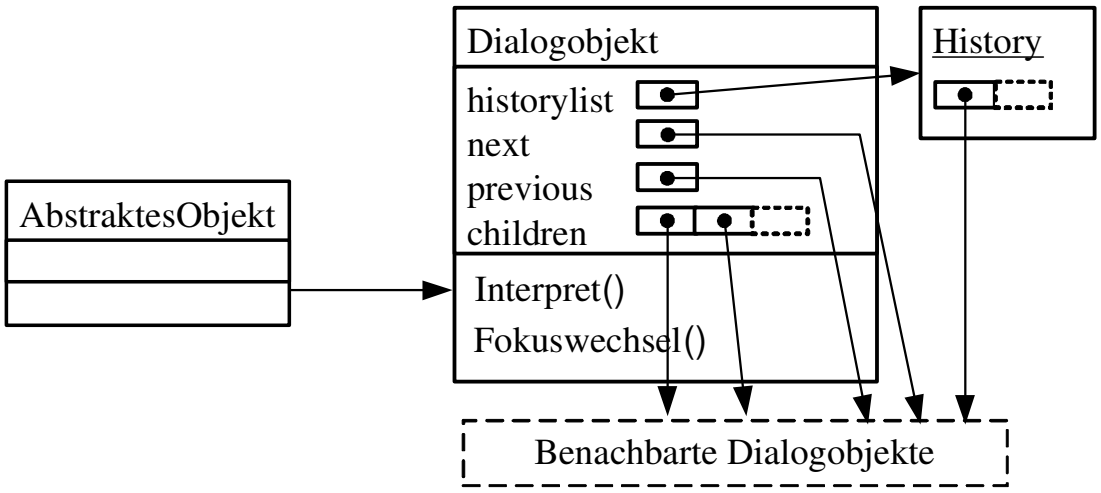

Abbildung 61: Die Ermittlung der benachbarten Dialogobjekte

Das über den Fokuswechsel benachrichtigte Dialogobjekt löscht zunächst die vorhandene Präsentation. Dann ermittelt es RoV-Parameter (aktuell nur RoV=1) und beginnt mit der Neudarstellung der Präsentation. Dazu präsentiert das Dialogobjekt sich selbst und fragt die Namensfelder seiner untergeordneten Dialogelemente (Childobjekte) ab. Bei RoV=1 wird lediglich die Schicht der direkt untergeordneten Elemente angesprochen. Nun kommen die Navigationselemente zur Darstellung, dazu wird das *next*- und das *previous*- Attribut des Dialogobjekts ausgewertet.

Als Bestanteil des Interpreters hat jedes Rendermodul Zugriff auf globale Daten des Interpreters, wie z.B. die Historyliste (Abbildung 62). In dieser Liste sind die zuletzt besuchten Dialogobjekte gespeichert. Aus der Liste wird nun das vorher besuchte Dialogobjekt ermittelt dessen Namensfeld ebenfalls präsentiert.

[21] Ein Fokuswechsel erfolgt, wenn der Benutzer ein Dialogobjekt verlässt und ein neues aktiviert.

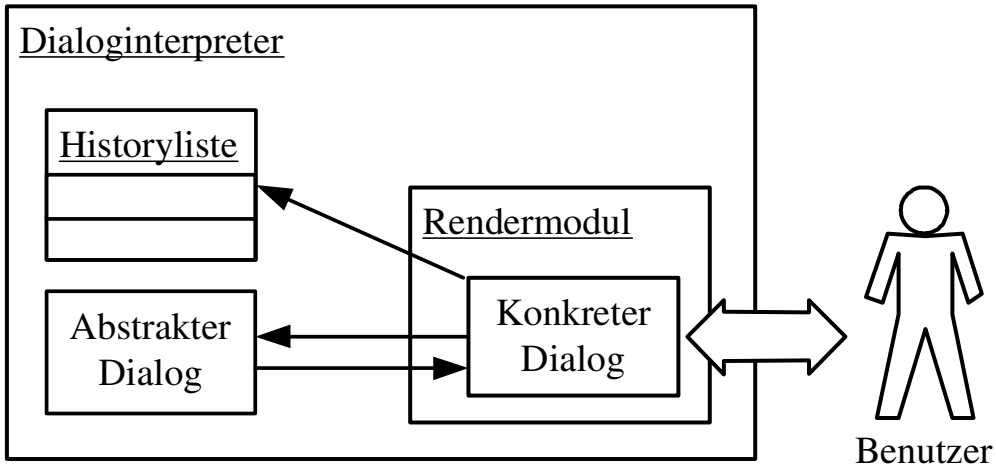

Abbildung 62: Zugriff auf die globale Historyliste

Um die mehrfache Darstellung gleicher Dialogobjekte zu eliminieren, wird in diesem Prozess jeweils geprüft, ob ein Element aktuell bereits dargestellt ist.

Das Ergebnis dieses Verfahrens ist die Begrenzung der Darstellung auf maximal ein Dialogobjekt. Auf die benachbarten Dialogobjekte wird lediglich verwiesen. Die Nachbarn sind durch Navigation erreichbar. Ist ein Dialogobjekt ein Eingabeobjekt, kann können Daten eingegeben werden, sonst ist nur die Navigation möglich.

Wenn bei der Darstellung der Verweise in künftigen Realisierungen auf die Unterdrückung der Zyklen geachtete wird, kann *RoV* auch auf größere Werte eingestellt werden. Die Einstellung des RoV ist dann eine parametrische Methode, um den benötigten Platz für die Darstellung eines Dialogs an die Anzeigeauflösung anzupassen.

5.8 Klassifikation von Modalitäten

Mit der Zielarchitektur ist es möglich, verschiedene Endgerätetypen zu unterstützen. Das Konzept greift zur Adaption auf Rendermodule zurück, welche bestimmte Kombinationen aus Modalitäten bedienen können. Dem entsprechend könnte ein Rendermodul eine WIMP-basierte Benutzungsoberfläche realisieren, die sich aus Tastatur, Maus und grafikfähigem Bildschirm zusammensetzt.

Um zu einem flexiblen System zu gelangen, wird ein Satz an Rendermodulen vorgesehen, der Basismodalitäten bereitstellt. Die zur Interaktion mit den Engeräten gebrauchten Modalitäten können dazu klassifiziert werden. Die Klassen beziehen sich auf bestimmte elementare Interaktionsformen, die so kombiniert werden können, dass jedes heute übliche mobile Endgerät bezüglich seiner Interaktionsschnittstellen unterstützt werden kann.

Folgende Abbildung 63 zeigt, wie eine solche Klassifikation durchgeführt werden kann. Die Modalitätsklassen können von Rendermodulen unterstützt und somit in das System aufgenommen werden.

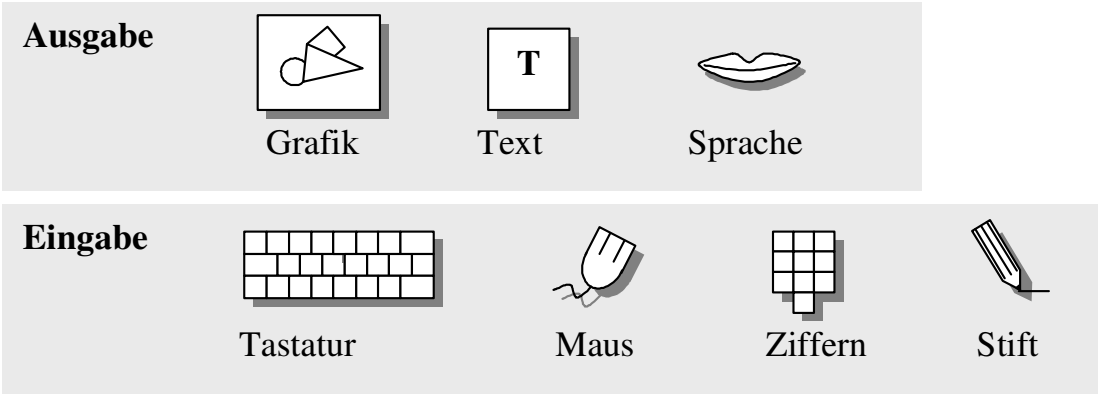

Abbildung 63: Klassifikation von Modalitäten

Interaktion kann als geräteunabhängig angesehen werden, wenn sie in Kombination mit *allen verfügbaren Endgeräten* konkret durchführbar ist. Deshalb ist es ausreichend, die Interaktionsschnittstellen heute verfügbarer mobiler Geräte zu betrachten. Mit einer einem Satz an Basismodalitäten kann nach Abbildung 64 jedes Endgerät durch Komposition unterstützt werden.

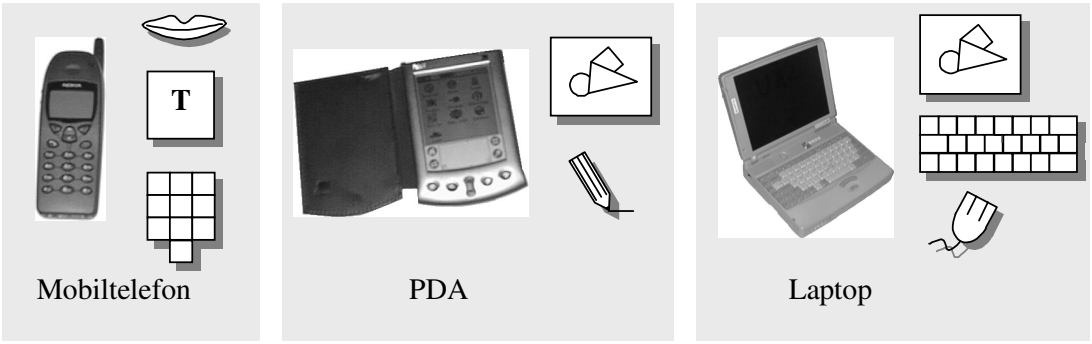

Abbildung 64: Die Abbildung von Interaktionsmethoden auf mobile Endgeräte

5.9 Verbindungserkennung

Bei natürlichen Dialogen wird eine Gesprächsverbindung durch den fortwährenden Austausch von gegenseitigen Signalen aufrechterhalten. In der Umkehrung führt das Ausbleiben dieser Signale zu einer Unterbrechung des Gesprächs, bzw. zum Gesprächsende.

Das mobile UIMS sollte dieses Verfahren analog benutzen, um die Gültigkeit der Verbindung zum Benutzer sicherzustellen. Andernfalls können unberechtigte Personen Zugriff auf Daten und Dienste des Benutzers erhalten. Bei Ausbleiben von Interaktionssignalen sollte ein Dialog entweder beendet oder in einen Wartezustand versetzt werden. Die Zeitdauer, nach der die Verbindung als unterbrochen gilt, wird für jeden Dialog festgelegt.

5.10 Beschreibungssprache

In Kapitel 2 werden Anforderungen an die Beschreibungssprache gestellt: Danach sollte die Beschreibungssprache die Interaktionsdaten strukturieren und die Interaktionsdaten abstrakt kodieren, so dass sie auf allen Zielgeräten darstellbar sind. Die Forderung nach Abstraktion bedeutet die Beschreibung von Dialogen ohne die Zugabe von gerätespezifischen Informationen oder ohne die Ausrichtung auf eine bestimmte Technologie.

5.10.1 Strukturierung

Als Basis für die Beschreibung dient das Dialogmodell aus Abschnitt 5.3. Es kommen alle Beschreibungssprachen in Betracht, mit denen Dialoge dieses Modells beschrieben werden können. Ein Dialog nach Abschnitt 5.3 ist charakterisiert durch

- zwei Gesprächspartner mit Namen (Agentenname←→Benutzername),
- durch einen Dialogbeginn, eine Dialogmitte und ein Dialogende. Der Dialoganfang bietet die Möglichkeit zur Identifikation und stellt eine Verbindung mit dem Benutzer her. Die Dialogmitte enthält Dialogschritte, die der Ein- und Ausgabe von Daten dienen. Das Dialogende dient der Ratifikation des Dialogabbruchs und bietet die Möglichkeit wieder in die Dialogmitte zu einem beliebigen Dialogschritt zu gelangen.
- durch Daten, die mit dem Mittel Text und Sprache dargestellt werden können. Nicht mit textuellen oder sprachlichen Mitteln darstellbare Inhalte (z.B. Bild) können referenziert werden, also Gegenstand eines Gesprächs sein, werden aber nicht weiter abstrahiert.

Die Dialogelemente weisen eine hierarchische Struktur auf und werden nach dem Konzept durch Verweise ergänzt. Die Verweise werden in die Attribute der Dialogelemente eingebracht.

Dialogelement	Attribute
Dialog	Name des Initiators
	Gültigkeitsdauer
	Dialogthema
Dialoganfang	Grußformel
Dialogschritt	Verweis auf einem anderen Dialogschritt
	Verweis auf den nächsten logischen Dialogschritt
	Titel des Dialogschritts
	Kurzbeschreibung des Dialogschritts
	Indikator, ob der Dialogsschritt unbedingt auszuführen ist
	Verweis auf ein beliebiges Objekt, dass Gegenstand des Dialogschritts ist
Eingabetyp	Vorgabewert
	Titel des Datenelements
	Kurzbeschreibung des Datenelements
	Indikator, ob die Eingabe unbedingt auszuführen ist
Dialogende	Verabschiedungsformel

Tabelle 5: Konzept für die Sprachelemente der Dialogbeschreibungssprache

Die Definition der Struktur benötigt Sprachelemente, welche mit den Dialogelementen assoziiert werden können. Für die Darstellung eines Dialogs genügen die Dialogelemente mit den jeweiligen Attributen aus Tabelle 5. Für die Beschreibung der Attribute werden Schlüssel-Wert-Paare benutzt.

5.10.2 Datentypen

Nun sollen die Datentypen besprochen werden, mit denen sich solche unspezifischen Dialoge beschreiben lassen Das Einlesen von Informationen durch den Rechner wird durch Ausgaben an den Benutzer eingeleitet, die ihn über den Zweck und die Art der Eingabe informieren. Bei grafischen Systemen können die Ausgaben impliziter Art sein, so Bedarf es z. B. keiner weiteren Erklärung, dass Texteingabefelder mit der Tastatur bedient werden. Sollte nun in einem anderen Fall die gleiche Eingabe rein sprachlich erledigt werden, muss dem Benutzer die Art der erforderlichen Eingabe mitgeteilt werden. Dialoge müssen mit solchen zusätzlichen Informationen versehen werden, die Auskunft über die Art der Eingabe geben, damit sie dem Benutzer Auskunft über die Optionen geben können. Wesentlich für die Dialogelemente ist die

Art der möglichen Eingaben, d.h. der Typ der Eingabeparameter. Hierbei lassen sich die folgenden Parametertypen identifizieren:

- Kommandos: Z. B. „OK“, „Weiter“, etc.
- Boolesche Werte.
- Ganze Zahlen (Integer)
- Fließkomma-Zahlen
- Zeichenketten: Text, auch Zahlenkolonnen, Daten (allgemein)
- Referenzen auf Mediendaten: Dateinamen, URLs, etc..
- Ereignisse: Navigation zwischen den Dialogelementen, Starten/Beenden eines Dialogs.

Besondere Beachtung verdienen Kommandos und die Steuerung bzw. Navigation in Dialogen mit zeitabhängigen Medien. Mit diesen Grundelementen lassen sich Dialoge nach Eingabeparametern grundsätzlich beschreiben. Nachfolgend sind einige typische Dialogelemente für diese Parameter zusammengestellt:

Technik	**Bool**	**Integer**	**Float**	**Text**	**Kommandos**
Tastatur	Tasten mit den beiden Optionen belegen	Tastenkom-binationen mit Optionen belegen	Tasten mit Ziffern belegen	Tasten mit Buchstaben belegen	Tasten mit Kommandos belegen
Sprache	Optionen anbieten, z. B. „Ja“ oder „Nein“	Optionen Anbieten, z. B. „A“,“B“ oder „C“	Diktieren/ Buchstabieren	Diktieren	Worte mit Kommandos belegen
Grafik (Maus/Pen)	z. B. mit Buttons, Checkbox, …	Radiobutton, stufiger Schieberegler	Schiebe- oder Drehregler	Handschrift, Bildschirm-tastatur	Kommando-element auswählen
Mimik/Gestik	2 Gesten mit 2 Optionen belegen	Ziffernweise, z. B. für Gesten von 1-5	Geste mit Prozentwerten oder Ziffern belegen	Gesten mit Buchstaben oder Begriffen belegen	Gesten mit Kommandos belegen

Tabelle 6: Eignung von Techniken für die Eingabe bestimmter Datentypen

5.11 Migration

Eine wesentliche Anforderung an das mobile UIMS ist die Migration laufender Dialoge von einem Endgerät auf ein anderes. Dies erfordert eine Möglichkeit Dialoge anzuhalten und den aktuellen Zustand für eine Übertragung zu speichern.

Das Konzept für die Migration ist in Abbildung 65 anhand eines Beispieldialogs mit den abstrakten Dialogobjekten A1, A2, A3 mit dem jeweiligen Zustand S1, S2, S3 dargestellt. Die Dialogobjekte seien in der Reihenfolge A1, A2, A3 ausgeführt worden.

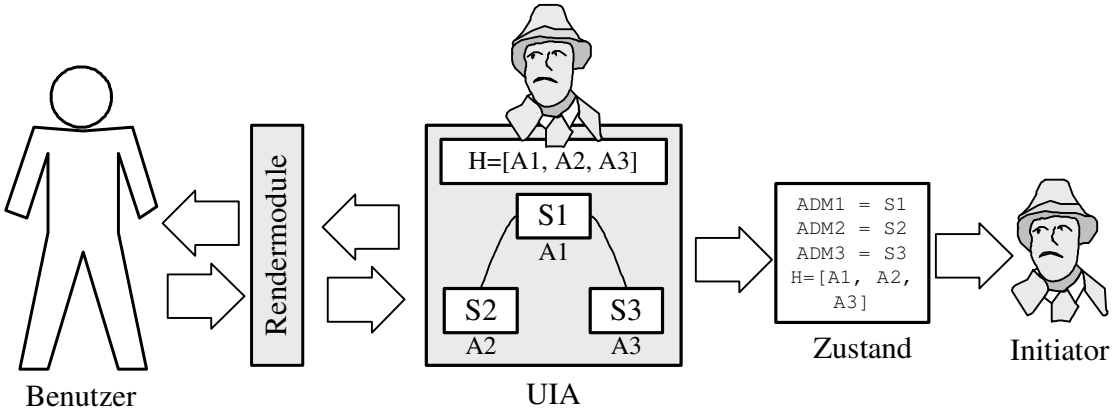

Abbildung 65: Der UIA übergibt den Dialogzustand an den Initiator

Bevor die Migration beginnt bekommt der UIA vom Benutzer oder vom Initiator die Anweisung für die Migration. Infolgedessen wird der Dialog durch den UIA angehalten und der Zustand der abstrakten Dialogelemente in einer Datenstruktur zusammengefasst. Außer dem Zustand der Dialogelemente wird der Dialogkontext in die Datenstruktur eingebracht Im Beispiel ist besteht ist der Kontext der Einfachheit halber die Ausführungsreihenfolge der abstrakten Dialogelemente. Der Initiator nimmt die Daten entgegen und migriert zum Zielsystem. Der UIA des Ausgangssystems schließt den Dialog.

Wenn der Initiator auf dem Zielsystem angekommen ist, meldet er sich dort beim lokalen Initiator an und beauftragt ihn mit der Weiterführung des Dialogs. Dazu übergibt er dem UIA den Ausgangsdialog und zusätzlich den Dialogzustand, den er auf dem Ausgangssystem entgegengenommen hat.

Die Wiederherstellung des Dialogkontexts kann Probleme bereiten, denn die Ausführbarkeit des Dialogs kann auf dem Zielsystem eingeschränkt sein. Dies ist dann der Fall, wenn eine bestimmte Eingabeform auf dem Zielsystem nicht verfügbar ist, aber zur Erreichung des Dialogziels benötigt wird.

Nun können alternative Reaktionen erfolgen:

- der Initiator kehrt unmittelbar zum Ausgangssystem zurück und meldet einen Fehler, bzw. nimmt die Dialogausführung dort wieder auf oder

- der UIA führt nur Teile des Dialogs aus und der Initiator verzögert die nicht zugänglichen Dialogteile, bis er sich auf einem geeigneten Endgerät befindet.

Weitere potentielle Fehlerfälle sind:

- Das Zielsystem könnte nicht verfügbar sein,
- auf dem Zielsystem befindet sich kein UIA,
- das Zielsystem fällt nach der Migration aus oder wird vom Netz getrennt, der Initiator geht verloren bzw. ist auf dem Zielsystem gefangen.

Das Konzept für den UIA soll die beiden erstgenannten Fälle berücksichtigen. Nach Möglichkeit soll der UIA einen Dialog darstellen, auch wenn Teile davon nicht ausführbar sind. Dazu soll die Dialogbeschreibung für jedes Dialogelement ein Attribut vorsehen, dass den UIA darüber informiert, welche Dialogelemente für die Ausführung zwingend benötigt werden.

5.12 Kommunikationskomponente

Die Kommunikationskomponente realisiert die Schnittstelle von den internen Komponenten des UIA zu den Initiatoren, die sich auf der Agentenplattform befinden. Die Aufgabe der Kommunikationskomponente besteht darin, die vorhandene Kommunikationsinfrastruktur zu adressieren und zu verwenden. Da der UIA nicht von einer bestimmten Agentenplattform abhängig sein soll, muss die Kommunikationskomponente im Nachhinein austauschbar sein, ohne dass Änderungen an anderen Komponenten des UIA notwendig werden.

Die Kommunikationsinfrastruktur kann verschiedenartig sein und die Agentenkommunikation auf verschiedenen Abstraktionsstufen bereitstellen. Das Spektrum reicht von der einfachen formatlosen Telegrammübertragung, über formatgebundene Sprachen mit Protokolldefinitionen, wie etwa FIPA-ACL, bis hin zu höherwertigen Sprachen, wie KQML oder FIPA-SL0. Die Austauschbarkeit der Kommunikationskomponente gewährleistet die Unabhängigkeit des UIA von der Verwendung einer bestimmten Sprache.

Der UIA ist dem Konzept zufolge ein stationärer Agent. Insofern kann die Kommunikation mit den Initiatoren einfach gehalten werden, da in jedem Fall vorausgesetzt werden kann, dass sich beide Kommunikationspartner auf dem selben Agentenserver befinden.

Die Kommunikationskomponente muss imstande sein, mehrere Kommunikationsprozesse gleichzeitig zu verwalten. Deshalb wird für jede Kommunikationsbeziehung zwischen UIA und Initiator eine eindeutige Identifikationsnummer vergeben. Diese Nummer bleibt so lange gültig, bis die Kommunikationsbeziehung zwischen Initiator und UIA endet. Mit der Forderung eines Initiators nach der Ausführung eines Dialogs beginnt eine Kommunikationsbeziehung. Mit dem Ende eines Dialogs wird dementsprechend auch die Kommunikationsbeziehung aufgelöst. Es ist nicht notwendig, den Initiator auf einen gleichzeitigen Dialog zu begrenzen, weshalb mehrere gleichzeitige Kommunikationsbeziehungen zwischen UIA und Initiator möglich sind.

Die Kommunikation zwischen Initiator besteht aus einzelnen Botschaften, die in ihrer Abfolge zu Protokollen zusammengefasst werden können. Ein Protokoll realisiert hierbei jeweils eine Funktion des UIA.

Die von der Kommunikationskomponente bereitgestellten Protokolle beinhalten alle Funktionen, die vom UIA angeboten werden. Diese Funktionen erlauben

- das Öffnen, Schließen und Abbrechen eines Dialogs.
- das Setzen eines oder mehrerer Attribute eines Dialogelements.
- das Melden einer Benutzereingabe an den Initiator.
- das Anhalten, die Abfrage des Status´ und das Weiterführen eines Dialogs.

Innerhalb des UIA kann die Kommunikation auf einer Low Level Schnittstelle erfolgen. Daher wird zu den internen Komponenten eine aufrufbasierte Schnittstelle vorgesehen. Zwar könnte diese Schnittstelle ebenfalls auf Basis einer Agentenkommunikationssprache erfolgen, doch wäre dann jede der Komponenten abhängig von der durch die Plattform vorgegebene Kommunikationsschicht.

Eine weitere Aufgabe der Kommunikationskomponente besteht in der Registrierung des UIA beim Directory Facilitator (DF) der Agentenplattform. Viele Agentenplattformen stellen diesen (Vermittlungs-)Agenten bereit. Der DF wird von Agenten dazu benutzt, einen bestimmten Dienst, der durch einen anderen Agenten in einer Agentenplattform bereitgestellt wird, zu finden. Wird der gesuchte Dienst von einem Agenten angeboten, kann eine Kommunikationsverbindung hergestellt werden. Ohne einen DF, müsste der UIA als Dienstanbieter im Agentensystem implizit bekannt sein.

5.13 Architektur für den User Interface Agenten

Die bisherigen Überlegungen werden nun zu einem Konzept für den UIA zusammengefasst, welches die Anforderungen aus Kapitel 2 erfüllen kann. Das Konzept wird in Form einer Architektur vorgestellt.

Das in Abbildung 66 dargestellte Blockschaltbild zeigt die Lösungsarchitektur für den UIA. Die Plattform für den unmittelbaren Betrieb des Agentensystems wird durch das Endgerät bereitgestellt. Außerhalb des UIA befinden sich die auftraggebenden Agenten (Initiatoren). Die Initiatoren fordern bei Bedarf die Eröffnung eines Dialogs, indem sie eine abstrakte Dialogbeschreibung an den UIA übergeben.

Die Kommunikationskomponente des UIA arbeitet einerseits mit dem Message Transport System der Agentenplattform zusammen und andererseits mit dem Dialogmanager des UIA.

Die Kommunikationskomponente nimmt die Botschaften der Agenten entgegen und extrahiert die Interaktionsdaten und übergibt sie an den Dialogmanager. Bei einem neuen Dialog übergibt der Dialogmanager die Dialogbeschreibung an das Parsermodul.

Das Parsermodul nimmt die Dialogbeschreibung entgegen und erzeugt daraus ein abstraktes Dialogmodell, welches an den Interpreter zur Ausführung übergeben wird. Zu dieser Transformation benutzt das Parsermodul die Grammatik, welche die Basis für die Prüfung der Syntax der abstrakten Dialogbeschreibung ist.

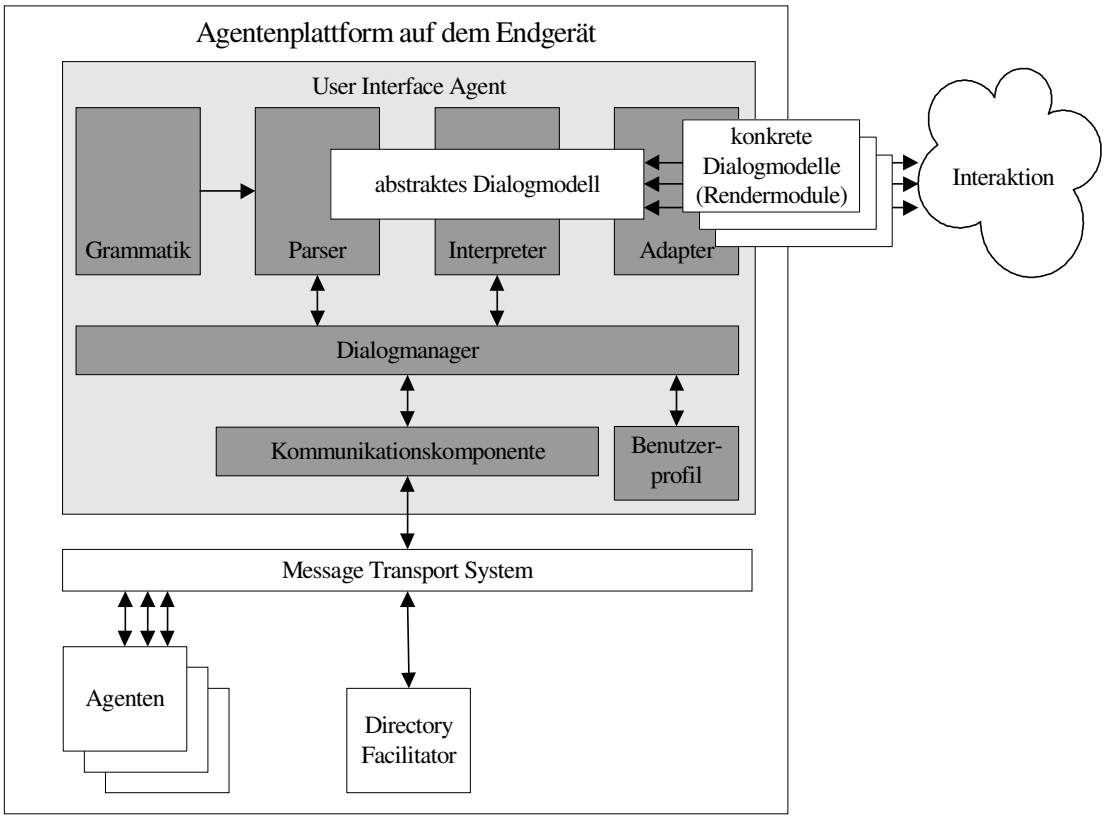

Abbildung 66: Architektur für den UIA

Der Interpreter nimmt das abstrakte Dialogmodell entgegen und fordert das Dialogmodell zu seiner Initialisierung auf. Bei der Initialisierung nehmen die abstrakten Dialogelemente mittels der Adapterkomponente Verbindung mit den Rendermodulen auf und fordern die Erzeugung ihrer konkreten Repräsentanten (konkrete Dialogelemente) an.

Des Weiteren steuert der Interpreter die Ausführung des Dialoges auf der abstrakten Ebene und hat Zugriff auf den Status der Interaktionselemente. Er nimmt die benutzerseitigen Ereignisse entgegen und leitet sie an den Dialogmanager weiter, der sie seinerseits an die Agenten meldet. Außerdem informiert der Interpreter die Dialogelemente über Ereignisse, die von den Agenten ausgelöst wurden. Bei Erreichen des Dialogendes ermittelt der Interpreter den Zustand aller Dialogelemente und kommuniziert diesen Status an den Initiator.

Das Benutzerprofil erlaubt die Konfiguration des UIA an die Erfordernisse und Wünsche des Benutzers. Hier kann festgelegt werden, welche Rendermodule auf dem Endgerät benutzt werden sollen.

5.14 Autorenumgebung

Nachdem die Architektur für den UIA erarbeitet wurde, soll im restlichen Teil des Abschnitts das Thema der Autorenumgebung Raum bekommen. Ohne dem Entwickler ein Mittel für die Erstellung und das Testen von mobilen Benutzungsoberflächen an die Hand zu geben, kann ein UIA nicht mit Anwendungen ausgestattet werden. Mit den bisher beschriebenen Systemkomponenten ist auch die Realisierung einer einfachen Autorenumgebung möglich.

5.14.1 Erstellung von Dialogbeschreibungen

Eine mobile Benutzungsoberfläche besteht zunächst aus der Beschreibung der Interaktionsdaten, die im vorliegenden Fall durch eine XML basierte Dialogbeschreibungssprache kodiert werden. Daher kann Beschreibung einer Benutzungsoberfläche mit einem einfachen Texteditor erfolgen, der inzwischen praktisch mit jedem Betriebsystem ausgeliefert wird. Diese Methode ist jedoch bei komplexeren Benutzungsoberflächen umständlich und deshalb fehleranfällig. Alternativ eine allgemeiner XML Editor benutzt werden, der die Struktur eines XML Dokuments bei der Darstellung berücksichtigt und die Eingabe eines XML Dokuments somit grafisch unterstützt.

5.14.2 Einbettung der Ereignisbehandlung in die Initiatoren

Einen weiteren Teil einer mobilen Benutzungsoberfläche stellen die Behandlungsfunktionen für die Interaktionsereignisse dar. Diese Methoden sind Teil der Initiatoren und müssen vom Programmierer der Agenten mit Funktionalität ausgefüllt werden. Die Einbettung der Ereignisse in einen Initiator-Agenten erfolgt durch simples Einfügen eines Code-Gerüsts, das der Entwicklungsumgebung beigegeben wird. Das Code-Gerüst gibt dem Programmierer Hilfestellung aufgrund eines einfach zu verstehenden Aufbaus.

5.14.3 Test der mobilen Benutzungsoberfläche

Die Präsentation einer Benutzungsoberfläche während des Erstellungsprozesses ist Hilfreich und kann dazu verwendet werden Funktionstests durchzuführen. Jedoch stellt eine konkrete Präsentation immer nur eine von vielen möglichen Varianten dar. Schließlich ist es eine Funktion des UIA die Präsentation einer Benutzungsoberfläche automatisiert an ein Endgerät

anzupassen und hierbei Konfiguration des jeweiligen Benutzers zu berücksichtigen. Die Präsentation einer in der Entwicklung befindlichen Benutzungsoberfläche wird durch einen einfachen Präsentationsagenten vorgenommen, der dem Entwickler erlaubt ein Dokument mit einer Dialogbeschreibung an den UIA zur Darstellung zu senden. Hierbei wird keine Änderung am UIA erforderlich.

5.15 Zusammenfassung

In diesem Kapitel wurde das Konzept für den UIA schrittweise entwickelt. Es nimmt Bezug auf die in Kapitel 2 formulierten Anforderungen und ist auf die Verwendung in einem mobilen Agentensystem ausgerichtet.

Die Kernaufgabe des UIA - nämlich die Adaption von verschiedenartigen Endgeräten - ist durch die Anwendung verschiedener Techniken erreichbar. Die Adaption erfolgt über Rendermodule, welche über eine Adapterkomponente mit dem System verbunden sind. Die Rendermodule führen den konkreten Teil eines Dialogs aus, der ebenfalls eine abstrakte Repräsentation besitzt. Um allen, durch die Geräte entstehenden Bedingungen, möglichst flexibel begegnen zu können, wurde für die Rendermodule eine Klassifikation vorgeschlagen. Diese Klassifikation soll den Zusammenbau von Benutzungsoberflächen aus generischen Rendermodulen ermöglichen.

Das System realisiert eine mehrstufige Synchronisation, entsprechend einem Pipeline Modell. Die Synchronisation erfolgt auf der abstrakten und der konkreten Ebene Elementweise. Die als Data-Fusion bezeichnete Zusammenführung der Benutzereingaben und die als Data-Splitting bezeichnete Verteilung der Systemseitigen Ausgaben erfolgt auf der abstrakten Ebene.

Durch die Mobilität der Dialogausführung kann es im Einzelfall zu Schwierigkeiten bei der Eingabe bestimmter Datentypen kommen. Die Eingabe eines langen Textes mit Hilfe einer Telefontastatur stellt keinen wünschenswerten Anwendungsfall dar. Um solche Fälle adäquat behandeln zu können, wurde die Signalisierung der Ein- und Ausgabefähigkeiten durch die Rendermodule in das Konzept mit aufgenommen. Die Migration ist mit dem vorgestellten Konzept ebenfalls problemlos möglich.

An eine Autorenumgebung zur Erstellung und zum Test von mobilen Benutzungsoberflächen wurde ebenso gedacht. Die Beschreibungsdaten für die Benutzungsoberflächen können mit einem XML Editor bearbeitet werden, für die Darstellung wird ein Präsentations-Agent bereitgestellt, der die Beschreibungsdaten zu Testzwecken mit Hilfe des UIA präsentiert und es wird dem UIA ein Codegerüst beigelegt, dass zur Ereignisbehandlung in den einen Initiatoragenten eingefügt wird.

6 Realisierung

Um das Konzept aus Kapitel 5 zu validieren, muss der User Interface Agent (UIA) konkret realisiert werden. Die in diesem Kapitel dargestellte Realisierung, wurde im Rahmen des Leitprojekts MAP vorgenommen. Dies bedingt, dass allgemein einsetzbare Teile mit projektspezifischen Teilen aus zusammenarbeiten müssen. So wurden im Rahmen des MAP-Projekts bestimmte Rendermodule realisiert, die in MAP gefordert waren und es wurde auf die in MAP genutzte Agentenumgebung zurückgegriffen. Der Nachweis der Realisierbarkeit des UIA ist von dieser Beschränkung der Allgemeinheit jedoch nicht berührt, da die Realisierung hinreichend die Wirkungsweise und Güte des Ansatzes wiedergibt.

6.1 Basisplattform

Mobile Anwendungsprogramme benötigen zu ihrer Ausführung eine Laufzeitumgebung. Die Aufgabe der mobilen Laufzeitumgebung ist es, auf den verschiedenartigen Endgeräten eine einheitliche Prozessumgebung zu bieten. Um eine solche einheitliche Prozessumgebung zu erreichen, existieren im Wesentlichen zwei Varianten:

1. *Virtuelle Maschine*
   Die virtuelle Maschine (VM) ist ein Prozess, der einen gedachten Rechner imitiert und dessen Befehlsatz beherrscht. Die VM arbeitet auf den Endgeräten und führt eigens für sie implementierten speziellen Programmcode aus. Der Programmcode wird als Bestandteil des Entwicklungsprozesses mit entsprechenden Compilern erzeugt.
2. *Interpreter*
   Ein Interpreter übersetzt die in einer Programmiersprache abgefassten Programmtexte während der Laufzeit. Dies ist etwa bei Skriptsprachen der Fall. Der Programmtext ist dem Anwender zugänglich und die Übersetzung belastet die CPU während der Laufzeit zusätzlich.

Für die Realisierung wird die verbreitete Java virtuelle Maschine (JVM) der Fa. Sun verwendet [JAVA]. Eine Java-Installation ist aktuell für alle gängigen Betriebssysteme erhältlich und bietet eine einheitliche Laufzeitumgebung für Java-Programme. Mobile Endgeräte mit knappen Ressourcen sind durch die Java2 Micro Edition (J2ME) der Fa. Sun gesondert unterstützt [J2ME].

Ein System für mobile Anwendungsprogramme sollte möglichst viele verschiedene Endgeräte adressieren. Als Basis für die Realisierung des UIA wird deshalb die Javaversion 1.2 als

Laufzeitumgebung ausgewählt. Auf der Basis von Java sind außerdem die meisten der modernen Agentenplattformen realisiert, die damit als Systemplattform in Frage kommen.

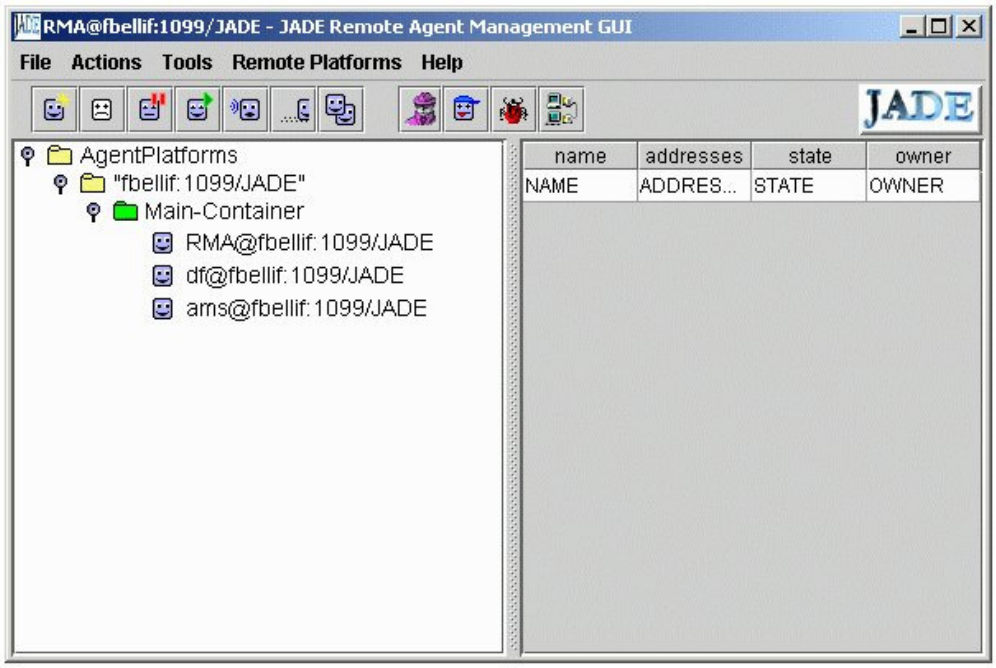

Abbildung 67: Die grafische Benutzungsoberfläche des JADE-Agentenservers [JADE]

Als Agentenplattform wird das *Java Agent Development Framework* (JADE) eingesetzt [JADE]. JADE bietet gegenüber anderen Agentenplattformen einige Vorteile. Es ist kostenlos verfügbar und gut dokumentiert. Die Agentenplattform ist mit der FIPA-Spezifikation konform und realisiert einen mobilen Botschaftsaustausch nach FIPA-ACL. Wie in Abbildung 67 gezeigt, kann die Plattform sowohl mit einer grafischen Benutzungsoberfläche zur Steuerung der Agenten betrieben werden, als auch in Form eines Hintergrundprozesses, der vom Anwender nicht direkt bedient werden muss. Die Benutzungsoberfläche erlaubt das Erstellen so genannter Container und die Erzeugung von Agenten in den Containern. Wie in obiger Abbildung zu sehen ist, findet in sich diesen Containern u.A. auch der Directory Facilitator (hier als df@fbellif:1099/JADE), der im nun folgenden Abschnitt eine Rolle spielen wird. Mobile Endgeräte werden durch JADE-LEAP gesondert adressiert. Für die Realisierung des UIA wurde die JADE-Version 3.0 eingesetzt.

6.2 Die Einbettung des UIA in das Agentensystem

In diesem Abschnitt ist dargestellt, wie der UIA in seine Umgebung eingebettet ist und wie er mit seiner Umgebung kommuniziert.

6.2.1 Registrierung

Die Schnittstelle zwischen UIA und den Agenten sollte einem bereits vertrauten Schema folgen. Dieses Schema ist bei der JADE-Agentenplattform durch die Agentenkommunikation in FIPA-ACL gegeben. Der Kontakt zwischen dem Initiator und dem UIA wird gemäß Abbildung 68 über den Directory Facilitator (DF) der JADE-Agentenplattform vermittelt (vgl. Abschnitt 3.5.3). Um den UIA über den DF finden zu können registriert der UIA seinen Interaktionsdienst nach Programmstart beim DF. Der UIA ist stationär auf einem Endgerät und benutzt eine abstrakte Interaktionsbeschreibungssprache für die Beschreibung von Benutzungsoberflächen. Für die Kommunikation mit den Initiatoren benutzt der Agent REQUEST-Protokolle nach [FIPA].

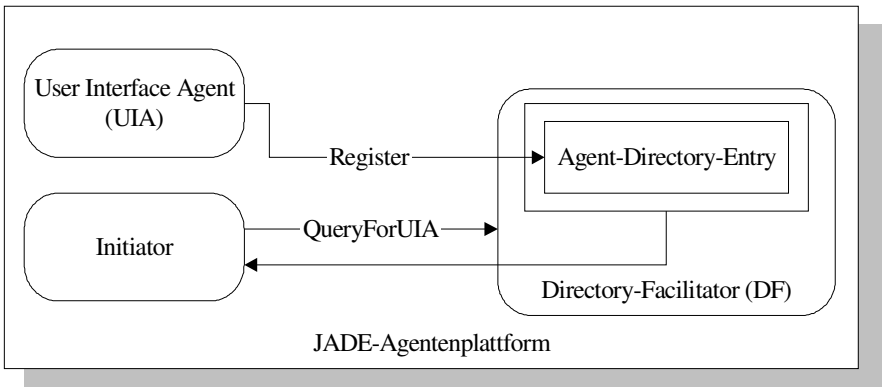

Abbildung 68: Der User Interface Agent registriert sich beim Directory-Facilitator

6.2.2 Agentenkommunikation

Nachdem der User Interface Agent sich beim DF registriert hat, können seine Interaktions-dienste von den anderen Agenten auf der Agentenplattform in Anspruch genommen werden. Die Inanspruchnahme der Dienste erfolgt über die Agentenkommunikation. Zur Eröffnung, Durchführung und Beendigung von mobilen Dialogen sind Protokolle realisiert. Die Protokolle folgen der FIPA-Spezifikation für Request-Protokolle [FIPAa]. Die Spezifikation der FIPA-

Protokolle weist jeder Botschaft eine *Performative* zu. Die Performative gibt Auskunft über die Funktion der Botschaft innerhalb eines Protokollablaufs.

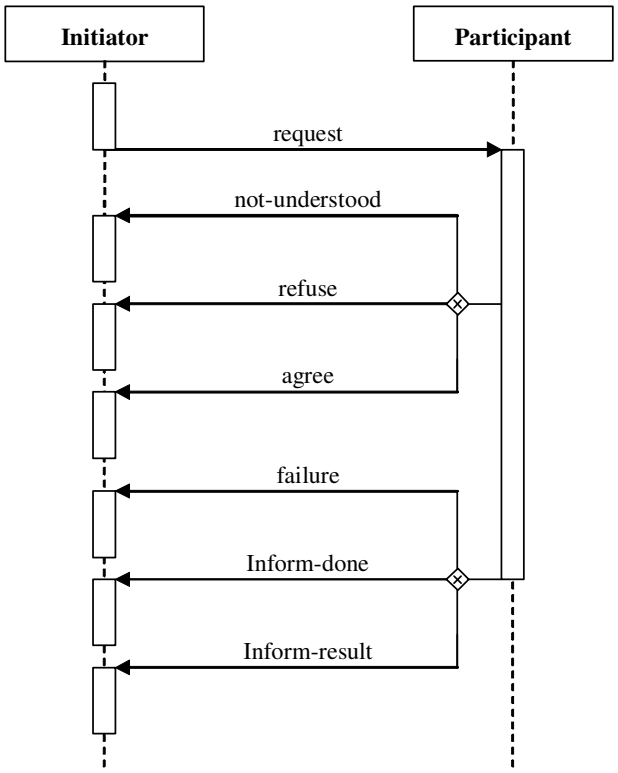

Abbildung 69: Das vollständige Request-Protokoll nach [FIPAa]

Der Protokollablauf eines Request-Protokolls nach FIPA beginnt entsprechend Abbildung 69 immer mit einer Request-*Performative*. Die vom Initiator gesendete Request-Botschaft kann durch eine von drei möglichen Performativen von *Participant* beantwortet werden:

1. *not-understood*
   Die Request-Botschaft wurde von *Participant* erhalten, aber sie wurde nicht verstanden. Hier kommen etwa inkompatible Basisformate oder Kodierungsfehler, etc. als Störquelle in Frage. In diesem Fall endet das Request-Protokoll hier.

2. *refuse*
   Die Request-Botschaft wurde von *Participant* erhalten und wurde verstanden, aber die gewünschte Funktion kann nicht ausgeführt werden. Dieser Fehler bezieht sich auf die Schicht der Anwendung und auf die Ausführbarkeit der angeforderten Funktion. In diesem Fall endet das Request-Protokoll hier.

3. *agree*
   Die Request-Botschaft wurde empfangen und verstanden und kann durch den Empfänger bearbeitet werden. Dies bezieht sich damit ebenfalls auf die Anwendungsebene.

Falls der angerufene Agent mit einer Agree-Botschaft antwortet, wird das Protokoll fortgesetzt. Als Folge kann der Initiator wiederum eine von drei Botschaften bekommen.

1. *failure*
   Bei dem Versuch die gewünschte Funktion auszuführen ist ein Fehler auf der Anwendungsebene aufgetreten.
2. *inform-done*
   Die Funktion wurde erfolgreich ausgeführt.
3. *inform-result*
   Die Funktion wurde erfolgreich ausgeführt und lieferte ein Ergebnis, welches Bestandteil dieser Botschaft ist.

Wenn der Initiator eine „inform-done"- bzw. eine „inform-result"-Botschaft erhält, wurde die Funktion erfolgreich ausgeführt.

Eine ACL-Botschaft der JADE-Plattform ist in sog. *Slots* aufgeteilt, die durch den UIA wie folgt benutzt werden:

- *receiver*
  Die Adresse des Agenten, der Botschaft die Botschaft empfängt.
- *performative*
  Die Performative der Botschaft. Sie wird als Zeichenkette kodiert. Beispiele: „request", „not-understood", „refuse", „agree" „information-done", etc.
- *content*
  Der Inhalt der Botschaft. Der UIA spezifiziert hier die auszuführende Funktion. Beispiel: „open dialog <CR> < XML-Beschreibung des Dialoges>".
- *conversationId*
  Dies ist eine Zeichenkette, welche die Kommunikationsbeziehung zwischen Initiator und UIA eindeutig bezeichnet. Die Zeichenkette beginnt mit dem Präfix „uia_req_" gefolgt von einer Nummer, die mit jedem neuen Dialog um 1 erhöht wird.
- *encoding*
  Dieser Slot enthält immer die Zeichenkette „plain-text"
- *ontology*
  Dieser Slot enthält immer die Zeichenkette „mobile uims"
- *language*
  Dieser Slot enthält immer die Zeichenkette „fipa-acl"

In nachfolgender Tabelle 7 sind alle Protokolle des UIA zusammengefasst.

Protokoll	Richtung	Beschreibung
Open Dialog	*I→UIA*	Das Öffnen eines neuen Dialogs durch den Initiator. Der Initiator bekommt eine eindeutige Identifikationsnummer für den Dialog zurück. Der UIA kann das Öffnen des Dialogs ablehnen, wenn das Medium oder der Kontext keinen neuen Dialog zulässt.
Cancel Dialog	*I→UIA*	Das Abbrechen eines laufenden Dialogs durch den Initiator. Die durchgeführten Eingaben werden verworfen. Das Abbrechen durch den Initiator kann die Folge einer Zeitüberschreitung o. Ä. sein.
Cancel Dialog	*UIA→I*	Das Abbrechen eines laufenden Dialogs durch den UIA. Die durchgeführten Eingaben werden verworfen. Der Abbruch durch den UIA geschieht i. A. als Folge einer Benutzereingabe.
Close Dialog	*I→UIA*	Das Beenden eines laufenden Dialogs durch den Initiator. Die durchgeführten Eingaben werden angenommen.
Close Dialog	*UIA→I*	Das Beenden eines laufenden Dialogs durch den UIA. Die durchgeführten Eingaben werden angenommen.
Update Dialog	*I→UIA*	Änderung der Attribute eines Dialogs. Betroffene Teile des Dialogs werden neu dargestellt.
Event	*UIA→I*	Die Mitteilung des UIA an den Initiator, dass der Benutzer eine Eingabe gemacht hat.

Tabelle 7: Die Request-Protokolle des UIA

6.3 Dialogbeschreibungssprache

In den Überlegungen in Abschnitt 5.10 wurde die Abstraktion und die Strukturierung der Interaktionsdaten von der Dialogbeschreibungssprache gefordert. Diese Anforderungen wird durch die Verwendung einer Teilmenge der XML-Sprache *UIML* (vgl. Abschnitt 4.2) erfüllt. UIML kombiniert die Vorteile von XML mit Funktionen für die Abstraktion von Interaktionsdaten.

6.3.1 Datenformat

Aus der Verwendung von XML folgt das Datenformat für die Kodierung der Dialogbeschreibung. Damit können die Interaktionsdaten von Entwicklern und Autoren unter Verwendung einfachster Texteditoren eingesehen und verändert werden. Für den UIA werden aufgrund des Konzepts nicht alle Funktionen der UIML benötigt. Es kommen deshalb nur der

Strukturierungsteil und der Attributierungsteil der UIML zum Einsatz. Aus dem eben gesagten und der Spezifikation der UIML folgt die Struktur einer UIA gerechten Dialogbeschreibung wie in Abbildung 70.

Durch die Definition von Regeln für die Anwendung von UIML und die Erweiterung bzw. Einschränkung der Spezifikation von UIML entsteht eine an die Dialoganforderungen angepasste Beschreibung.

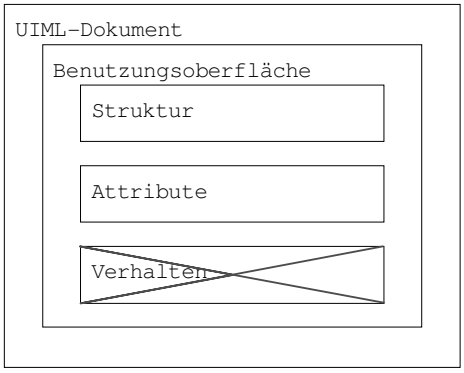

Abbildung 70: Die Struktur einer Dialogbeschreibung

Die XML-Spezifikation gliedert alle Dokumente durch die Verwendung von bestimmten Zeichenfolgen, die zur Markierung eingesetzt werden. Diese *Markups* oder *Tags* treten immer als Paar auf, das einen Bereich im Dokument umschließt. Der umschlossene Bereich wird im Folgenden als *Element* bezeichnet, das nach dem Markup benannt wird. So schließt etwa das *uiml-Element* ein den UIML spezifischen Teil eines XML-Dokuments ein. Die Elemente *interface*, *structure* und *style* schließen gemäß Abbildung 70 die Deklaration der Benutzungsoberfläche, ihren Strukturteil und ihren Attributierungsteil ein.

6.3.2 Sprachelemente

Das **interface-Element** beschreibt alle in einem Dokument enthaltenen Interaktionsbausteine. Dieses Element ist in die beiden Elemente *structure* und *style* aufgeteilt, so dass sich im Dokument folgender Aufbau ergibt:

```
<interface>
     <structure>…</structure>
     <style>…….</style>
</interface>
```

Das **structure-Element** definiert alle Teile der Benutzungsoberfläche und legt deren Anordnung fest. Die Teile einer Benutzungsoberfläche werden durch das nachfolgend dargestellte part-Element definiert. Jedes Teil hat eine Klasse und einen Dokumentenweit eindeutigen Bezeichner mit dem es referenziert werden kann.

Das **part-Element** wird verwendet, um ein Teil in die Benutzungsoberfläche einzufügen. Ein part-Element erwartet den Parameter *class*, der die zu verwendende Klasse angibt und den Parameter *name* für den Bezeichner. Eine die Definition eines Parts gestaltet sich etwa folgendermaßen:

```
…
<part class="Klassenname" name="Bezeichner">…</part>
…
```

Dieses Codefragment erzeugt ein Element der Klasse *Klassenname*, das mit der Zeichenkette *Bezeichner* referenziert werden kann. Auch die Schachtelung der part-Elemente ist möglich durch den folgenden Konstrukt, der die äußere Klasse *Aussen* mit der inneren Klasse *Innen* schachtelt:

```
<part class="Klassenname" name="Außen">
    …
    <part class="Klassenname" name="Innen">…</part>
    …
</part>
…
```

Durch die Schachtelung der part-Elemente wird eine hierarchische Gliederung des Gesamtdialogs ermöglicht. Jedes part-Element kann wiederum weitere part-Elemente enthalten, dadurch sind beliebig viele Gliederungsebenen möglich.

Das **style-Element** kennzeichnet den Teil der Interface-Beschreibung, in dem die Zuweisung von Eigenschaften an die im structure-Element eingeführten *Parts* vorgenommen wird. Dazu enthält das style-Element eine Anzahl von property-Elementen.

Das **property-Element** dient der Zuweisung eines Wertes an ein Attribut. Das mit der Eigenschaft zu versehende Part-Element wird durch seinen Bezeichner identifiziert. Der Bezeichner referenziert hierbei ein Part-Element, das im *structure*-Element eingeführt wurde. Das folgende Codefragment weist dem Attribut *Attributname* des Elements *Bezeichner* den Wert *Eigenschaft* zu.

```
<style>
    …
    <property part-name="Bezeichner" name="Attributname">
    Eigenschaft
```

```
        </property>
        …
    </style>
    …
```

6.3.3 Dialogklassen

Im vorangegangenen Abschnitt sind Möglichkeiten zur Beschreibung von Elementstruktur und Elementeigenschaften beschriebenen. Diese Methoden können auf die im Folgenden dargestellten Klassen angewendet werden.

DialogOpen repräsentiert die Dialogeröffnung und hat folgende Attribute:

Attributname	Typ	Beschreibung
whoami	String	Der Namen des Initiatoragenten
hello	String	Willkommenstext zur Begrüßung des Benutzers
next	String	Bezeichner des nächsten Dialogschritts

Tabelle 8: Attribute der DialogOpen-Klasse

DialogMove repräsentiert einen Dialogschritt aus der Dialogmitte. Die damit erzeugbaren Elemente können einerseits als Container für untergeordnete Dialogschritte dienen, andererseits können sie Eingaben vom Benutzer entgegennehmen. DialogMove hat gemäß Tabelle 9 die folgenden Attribute:

Attributname	Typ	Beschreibung
title	String	Der Titel des Dialogschritts
description	String	Beschreibung des Dialogschritts
mandatory	Y/N	Legt fest, ob dieser Dialogschritt ausgeführt werden muss, um das Dialogziel zu erreichen
priority	Integer	Legt die Priorität des Dialogschritts fest (1=niedrig, 2=mittel, 3=hoch)
input-type	Typ-String	Legt fest, ob der Dialogschritt ein Eingabetyp ist. Erlaubte Werte sind: boolean, integer, float, char, string, stringlist. Wenn input-type nicht gesetzt wird, ist dieses Element kein Eingabetyp
previous	String	Bezeichner des vorhergehenden Dialogschritts
next	String	Bezeichner des nächsten Dialogschritts

Tabelle 9: Attribute der DialogMove-Klasse

Zu den in Tabelle 9 aufgeführten Attributen kommen abhängig vom Eingabetyp weitere hinzu, wenn das Attribut *input-type* gesetzt ist. Diese zusätzlichen Attribute sind in Tabelle 10 gezeigt.

Attributname	Typ	Beschreibung
input-default	Bool/Integer/ float/String	Legt den Ausgangswert des Dialogschritts fest, falls dieser einen Eingabetyp hat
input-min	Integer/float	Legt den Minimalwert für eine gültige Eingabe fest bei *input-type=integer* oder *type=float*
input-max	Integer/float	Legt den Maximalwert für eine gültige Eingabe fest bei *input-type=integer* oder *type=float*
input-items	String[]	Liste von Ausgangswerten für *input-type=Stringlist*. Die Werte werden in Anführungszeichen gesetzt und durch Komma getrennt

Tabelle 10: Attribute der Eingabeklassen in Abhängigkeit von input-type

DialogEnd repräsentiert das Dialogende und hat die in Tabelle 11 dargestellten Attribute.

Attributname	Typ	Beschreibung
bye	String	Text zur Verabschiedung des Benutzers
previous	String	Bezeichner des vorhergehenden Dialogschritts. Hier wird der Bezeichner des obersten DialogMove-Elements der Dialogmitte eingetragen.

Tabelle 11: Die Attribute der DialogEnd-Klasse

6.3.4 Tokenizer und Parser

Der Tokenizer verfügt über die Definition der Grammatik. Anhand der Grammatik versucht er das gerade zu bearbeitende UIML-Dokument in Tokens zu überführen und diese in einer Liste zu speichern. Tokens stellen die Übersetzungen der Schlüsselworte bzw. Anweisungen aus dem UIML-Dokument in Objekte dar. Die Token-Objekte enthalten die Information über die Zeilennummer, das identifizierte Wort und den Typ des Wortes. Nach fehlerfreier Umsetzung des UIML-Dokuments stehen die Tokens in einer Liste gespeichert dem Parser zur Verfügung.

Der Parser hat Zugriff auf eine Klassenbibliothek und wandelt die Tokens aus der Liste in Objekte um. Die aus den Klassen erzeugten Objekte werden in einem hierarchischen Graphen gespeichert. Die Hierarchie des Graphen entspricht der Gliederung des UIML-Dokuments. Der Objekt-Graph ist die Basis für die Interpretation des Dialogs.

6.4 Schnittstelle zu den Rendermodulen

Für die Anbindung verschiedener Endgeräte mit unterschiedlichen Plattformen und Mechanismen zur Benutzerinteraktion wird eine universelle Schnittstelle benötigt. Diese Schnittstelle steht am Übergang zwischen abstrakter und konkreter Darstellung der Dialogausführung. Zur Verwirklichung der Adaption im geforderten Sinne empfiehlt entsprechend Abschnitt 5.6.1 die Anwendung des Abstract Factory Pattern nach [Busc96].

Für den User Interface Agent wird für ein Rendermodul die Implementation von 11 Javaklassen benötigt. Der Entwicklungsumgebung für den UIA liegt ein entsprechendes Framework aus leeren Klassen bei, welches vom Entwickler eines neuen Rendermoduls lediglich modifiziert werden muss. Im Folgenden wird Bezug auf dieses Framework genommen und es werden die Klassennamen aus dem Framework verwendet.

Das Framework besteht aus folgenden Klassen:

1. *renderFactory*
   Dies ist die abstrakte Factory-Klasse, die alle anderen Klassen instantiieren kann.
2. *renderDialog*
   Diese Klasse repräsentiert den Dialog im Rendermodul.
3. *renderOpeningElement*
   Diese Klasse repräsentiert die Dialogeröffnung im Rendermodul.
4. *renderMoveElement*
   Diese Klasse repräsentiert einen Dialogschritt im Rendermodul.
5. *renderEndElement*
   Diese Klasse repräsentiert das Dialogende im Rendermodul.
6. *renderBoolElement*
   Diese Klasse repräsentiert die Eingabe eines Booleschen Wertes im Rendermodul.
7. *renderIntegerElement*
   Diese Klasse repräsentiert die Eingabe einer ganzen Zahl im Rendermodul.
8. *renderFloatElement*
   Diese Klasse repräsentiert die Eingabe einer Fließkommazahl im Rendermodul.
9. *renderCharacterElement*
   Diese Klasse repräsentiert die Eingabe eines Textzeichens im Rendermodul.
10. *renderStringElement*
    Diese Klasse repräsentiert die Eingabe Textes im Rendermodul.
11. *renderStringlistElement*
    Diese Klasse repräsentiert die Auswahl einer Option im Rendermodul.

Entsprechend Abschnitt 5.6.2 besitzen außer dem Factory-Objekt und dem Dialog-Objekt, alle Dialogobjekte folgende Methoden:

- Konstruktor,
- `interpret()`
  fordert die Ausführung der Präsentation des Elements an.
- `update(<Spezifikation des Attributs>)`
  fordert die Aktualisierung eines Attributs und seine Anzeige an.
- `release()`
  fordert das Element an das Ende des Dialogs anzuzeigen.
- `inputOccurred(<Spezifikation der Eingabe>)`
  signalisiert eine Benutzereingabe auf einem anderen Rendermodul.
- `focusChanged(<Spezifikation des Fokuswechsels>)`
  signalisiert diesem Element einen Fokuswechse.

Im Mittelpunkt des Frameworks steht die abstrakte Klasse `abstractFactory`, die das Interface für alle Factory-Klassen der Rendermodule bereitstellt. Die Factory-Klassen der Rendermodule können instantiiert und zum Erzeugen von Dialogelementen benutzt werden (Abbildung 71). Bei der Erzeugung der Dialogelemente wird jeweils eine Methode `create(…)` aufgerufen, die entsprechende Objekte erzeugt und den Konstruktor des Objekts aufruft.

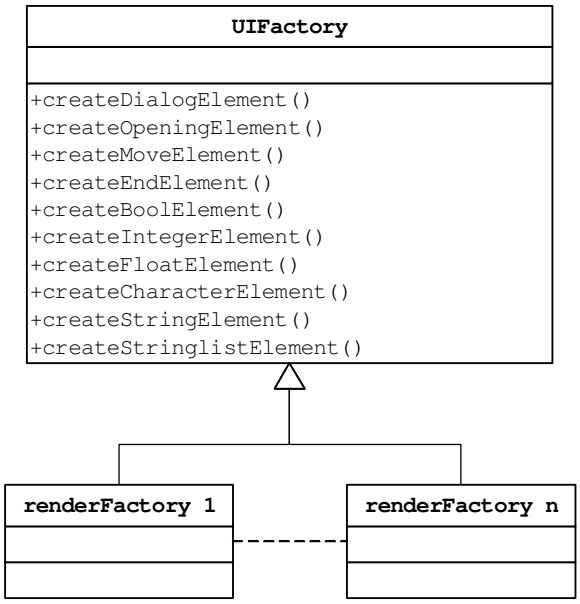

Abbildung 71: Die Implementierung der Rendermodule

6.5 Daten- und Kontrollfluss

Abbildung 72 zeigt das Zusammenspiel der Komponenten des UIA. Die Kommunikationskomponente holt die Botschaften vom Message Transport System der JADE-Plattform mittels einer `receive(…)`–Methode ab, bzw. verschickt Botschaften mit Hilfe der `send(…)`-Methode.

Einkommende Botschaften werden zunächst auf einen Botschaftsstapel gelegt, wo sie von der Kommunikationskomponente der Reihe nach abgeholt werden können. Die Kommunikationskomponente wickelt die FIPA-Request-Protokolle des UIA ab und übernimmt gleichzeitig Steuerungsaufgaben auf der Task-Ebene. So prüft sie etwa die Botschaften darauf, ob ein neuer Dialog zu öffnen ist, oder ob ein Event vom Initiator gesendet wurde. Die Kommunikationskomponente erzeugt bei einer Dialogeröffnung einen neuen Dialogprozess und leitet die in der Botschaft enthaltene Dialogbeschreibung an den Tokenizer weiter. Hingegen werden bei einem update-event die aus der Botschaft extrahierten Interaktionsdaten an den Interpreter weitergeleitet. Vom Benutzer ausgehende Interaktionsereignisse werden durch den Interpreter an die Kommunikationskomponente übergeben und anschließend an den Initiator gesendet.

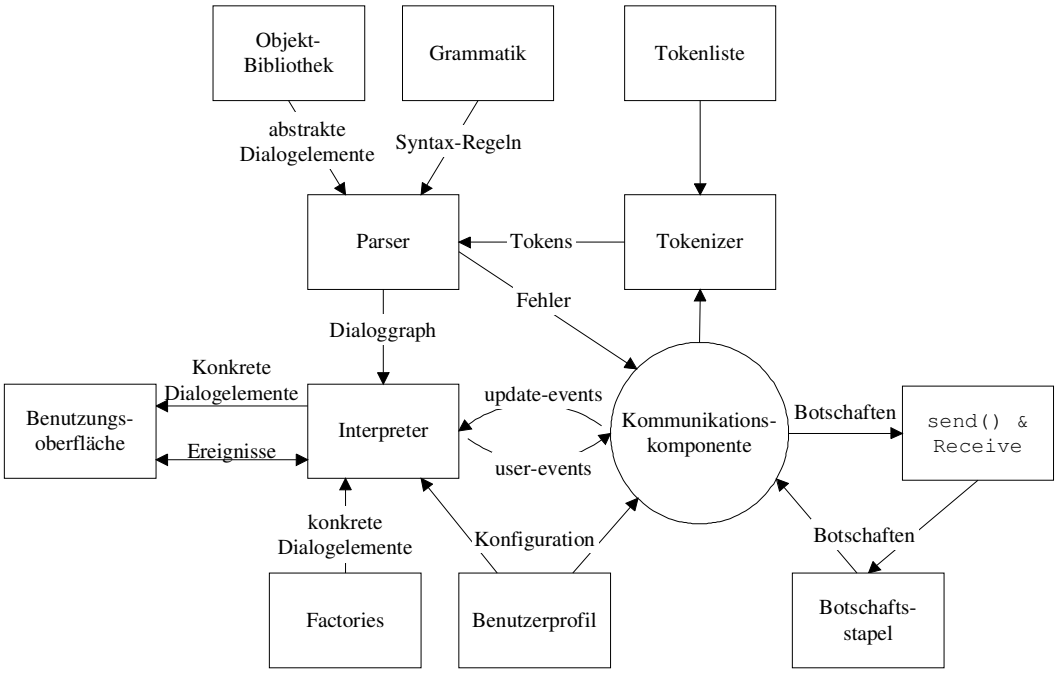

Abbildung 72: Daten- und Kontrollfluss im User Interface Agent

Der Tokenizer wandelt mithilfe einer Tokenliste die Dialogbeschreibung in Tokens um, so dass sie vom Parser mit Hilfe einer Grammatik und einer Objektbibliothek in einen Dialoggraphen umgewandelt werden können. Schlägt diese Transformation fehl, meldet der Parser dies der Kommunikationskomponente, die das damit verbundene Request-Protokoll mit einem Fehler beendet und den dazugehörigen Taskprozess terminiert.

Wenn der Parser die Transformation erfolgreich durchführen kann, wird der noch abstrakte Dialoggraph an den Interpreter übergeben. Der Dialoggraph stellt ein gemäß der Dialogbeschreibung strukturiertes Objektmodell dar, das vom Interpreter ausgeführt werden kann. Der Interpreter beginnt, indem er den Konstruktor des abstrakten Dialog-Objektes aufruft, was die Initialisierung der gesamten abstrakten und konkreten Dialogelemente zur Folge hat. Dazu gebraucht der Interpreter die Factory-Objekte der Rendermodule. Nach der Initialisierung wird das abstrakte Element für die Dialogeröffnung ausgeführt und damit auch die konkreten Elemente für die Dialogeröffnung. Im Übrigen setzt sich dieses Schema immer in gleicher weise fort: Der Interpreter ruft eine Methode eines abstrakte Elements auf und das abstrakte Element ruft wiederum seine konkreten Repräsentanten auf.

Die Ausführung der abstrakten Dialogobjekte führt zu der Präsentation der konkreten Dialogobjekte. Bei einer Präsentation könnne Ereignisse auftreten, die in den abstrakten Dialogobjekten erfasst werden. Die Ereignisse werden zur Synchronisation an alle betroffenen Dialogobjekte gemeldet. Gleichzeitig werden die susammengeführ (Data-Fusion) und dem Initiator mitgeteilt.

Das Benutzerprofil wird bei Start des UIA ausgelesen. Entsprechend der Vorgaben lädt der User Interface Agent die geforderten Factory-Objekte aus den Rendermodulen, um bei einem Dialog daraus konkrete Dialogelemente zu generieren. Des Weiteren hat das Profil Einfluss darauf, wie viele Dialoge der User Interface Agent simultan ausführt.

6.6 Dialogereignisse

In diesem Abschnitt ist dargestellt, welche Ereignisse während eines Dialogs auftreten. Diese Ereignisse und Aktionen beziehen sich auf die Aspekte Inhalt, Steuerung und Status.

- Kontakt zu Benutzer ist hergestellt. (Gesprächsbeginn, Identifikation, Ratifikation)
- Kontakt zum Benutzer ist verloren gegangen. (Kontext, erkennbare Ursache?)
- Der Benutzer hat den Dialog abgebrochen oder beendet.
- Der Benutzer hat das Dialogelement gewechselt.
- Der Benutzer hat eine Eingabe gemacht.
- Der Benutzer hat ein Kommando zur Steuerung eingegeben.
- Dem Benutzer soll eine Statusmeldung angeboten werden.

- Die Informationen, auf die sich ein Dialogelement bezieht haben sich verändert und muss neu angezeigt werden.
- Der Benutzer ist zu einem anderen Dialog gewechselt.
- Der Benutzer hat diesen Dialog wieder aufgenommen.
- Es ist ein Fehler (unvorhergesehener Zustand) eingetreten.

6.7 Benutzerprofil

Die benutzerabhängige Konfiguration des UIA erfolgt über ein Verzeichnis „uiaprofile“, das für jeden konfigurierbaren Parameter des UIA eine Datei enthält. Aktuell sind die drei Dateien *Renderer*, *noOfconcurrentDialogs und debug* im Profilverzeichnis untergebracht. Die Dateien haben die folgende Funktion:

- *Renderer*
  Diese Datei enthält Informationen über die zu verwendenden Rendermodule. Beim Start des UIA wird diese Datei ausgelesen und es wird für jedes in der Datei angegebene Rendermodul das entsprechende Factory-Objekt geladen, das während des Öffnens eines Dialogs konkrete Dialogobjekte erzeugt. Derzeit können folgende Rendermodule in die Datei eingetragen werden:
  - *„graphical“*
    Dies ist ein Rendermodul, das ein grafisches WIMP-Interface realisiert.
  - *„speechout“*
    Dies ist ein Rendermodul, das lediglich Sprachausgaben vornimmt. Eingaben kann dieses Modul nicht entgegennehmen
  - *„pico“*
    Dies ist ein experimentelles Rendermodul, das ein möglichst kleines WIMP-Interface erzeugt.
  - *„chat“*
    Dies ist ein experimentelles Rendermodul, das ein Menübezogenes Interface nur mit Textein- und –ausgaben erzeugt, wie es in einem Chat-System benutzt werden kann.
  - *„htmlplus“*
    Dies ist ein multimodales Rendermodul, das auf der Basis einer HTML-Erweiterung über das Internet arbeitet.
- *noOfconcurrentDialog*
  In dieser Datei ist festgehalten, wie viele Dialoge der User Interface Agent maximal zur gleichen Zeit ausführen soll. Wenn mehr Dialoge als festgelegt angefordert werden, beantwortet der User Interface Agent die Anfrage mit einer „refuse“ Botschaft. Das Minimum ist 1, das Maximum ist beliebig.

- *debug*
  Der Inhalt dieser Datei gibt Auskunft darüber, ob während der Laufzeit sog. Debug-information ausgegeben werden sollen. Mögliche Werte sind „true“, für Informationen ausgeben oder „false“, für keine Informationen ausgeben.

Zeilen die in den Dateien mit einem „#“-Zeichen beginnen, werden als Kommentarzeilen behandelt und überlesen.

6.8 Einsatz der Realisierung in MAP

Das MAP Projekt startete als Leitprojekt des Bundesministeriums für Wirtschaft und Technologie im Themenfeld Mensch-Maschine Interaktion in der Wissensgesellschaft. Das Projektkonsortium bestand aus 15 Partnern aus dem wissenschaftlich-institutionellen Bereich und aus Partnern aus der Industrie. Das Projekt begann im Sommer 2000 und dauerte drei Jahre lang. Das Projektergebnis bestand in einer Plattform zur Kommunikation und Kooperation verschiedener Agentenplattformen, die in repräsentativen Arbeitsszenarien eingesetzt und getestet wurden. Die Schwerpunkte der Forschung bestanden in

- der Einbindung von mobilen Endgeräten, die mit multimodalen Schnittstellen ausgestattet sind. Diese wurden dazu benutzt, um die Bedienbarkeit der Benutzungsoberflächen durch die Verknüpfung der verschiedenen Interaktionsmodalitäten zu optimieren.
- der Entwicklung von agentenbasierten Anwendungen, die sich der Sicherheitsproblematik in verteilten Netzwerken annehmen.
- dem Nachweis der Tragfähigkeit der in MAP umgesetzten Ansätze. Dies wurde durch die Konstruktion von repräsentativen Szenarien, welche die gebauten Prototypen einsetzten, erreicht. Die Benutzertests wurden anhand von Szenarien durchgeführt.

Idee und grundsätzlicher Anspruch des MAP-Systems ist die Erweiterung der Informationstechnologie um Assistenz- und Delegationsfunktionen. Dieser Anspruch folgt aus einem propagierten, nun anstehenden Paradigmenwechsel bei Arbeitsplatzsystemen, der Weg von netzwerkbezogenen Einzelanwendungen hin zu Assistenz- und Delegations-Gruppenanwendungen in integrierten Netzwerken führt, wie in Abbildung 73 dargestellt.

MAP realisiert solche Dienste durch stationäre PC-Systeme und mobile Computersysteme, die zu einem Netzwerk verbunden sind.

Die Erweiterung um Assistenz- und Delegationsfunktionen stellt sich als Paradigmenwechsel in der Benutzungsoberfläche dar. So tauchen im MAP System zusätzlich zu den bekannten WIMP Oberflächen, die auf direkter Manipulation einzelner, meist visuell dargestellter Objekte beruhen, nun auch konversational ausgelegte Interfacemethoden auf [WKL01].

Die Integration von intelligenten Assistenzsystemen hat die Übernahme von Routinearbeiten des Benutzers zum Ziel. Die Orientierung der Mensch-Maschine Interaktion an menschlichen Umgangsformen sowie die Berücksichtigung von Mobilität und die Etablierung von

Sicherheitsmaßnahmen strebt die Optimierung des Arbeitsprozesses an, um eine Entlastung des Nutzers und eine Steigerung der Attraktivität des Arbeitsumfeldes zu erreichen.

Abbildung 73: Die Entwicklung der Arbeits- und Nutzungsformen von Büro-EDV [KP01]

Ein Ziel des Projektes war daher die Entwicklung von Software-Basiskomponenten zur Realisierung mobiler multimedialer Arbeitsplätze, die in unterschiedlichsten Anwendungen genutzt werden können. Die neben den Basiskomponenten in diesem Leitprojekt realisierten prototypischen Arbeitsplätze dienen dazu, neue Potentiale gezielt zu untersuchen, sie zu bewerten und entsprechend auszubauen. In einem Feldversuch wurde die Effektivität, Akzeptanz und Wirtschaftlichkeit dieser neuartigen Systeme überprüft. Damit bildet MAP einen idealen Ausgangspunkt für die Validierung des in dieser Arbeit vorgestellten Konzepts zur mobilen Mensch-Maschine Interaktion.

6.8.1 MAP-Systemaufbau

Das MAP System basiert auf vier technologischen Arbeitsbereichen, die sich in vier entsprechenden Teilprojekten (TP) niedergeschlagen haben. Das Zusammenwirken dieser Teilprojekte ergeben die MAP-Anwendungen, die mobile Arbeitsszenarien für die Planung und das Management im Bauwesen realisiert haben (TP6). Die Untersuchungen, welche in einem gesonderten Bereich (TP5) durchgeführt wurden, beziehen sich auf diese Anwendungen und schließen damit alle Technologiebereiche ein. Die Projektstruktur ist in Abbildung 74 zu sehen:

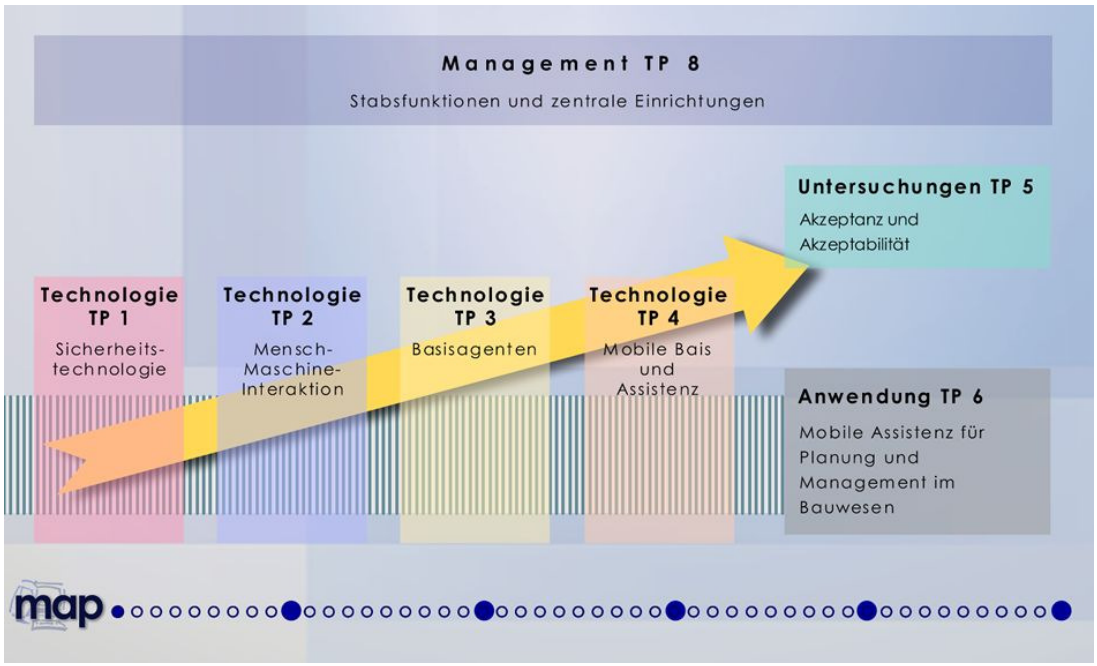

Abbildung 74: Die Gliederung des MAP-Projekts [KP01]

Die vier Technologiebereiche haben folgende Aufgaben:

- *Sicherheitstechnologie*:
  Entwicklung von Systemen und Mechanismen zur Datensicherheit und Datenschutz, der Schutz des Einzelnen vor Missbrauch persönlicher Daten (informationelles Selbstbestimmungsrecht) und Verschlüsselung der Kommunikationskanäle (kommunikatives Selbstbestimmungsrecht). Darüber hinaus sicherheitsrelevante Themen wie der wechselseitige Schutz von Agenten vor böswilligen Servern, sowie die Absicherung von Servern vor virulenten oder spionierenden Agenten.
- *Mensch-Maschine Interaktion:*
  Bereitstellung einer Infrastruktur, die Aufgaben auf stationären und mobilen Endgeräten erledigen kann. Durch die optionale Verwendung eines Conversational User Interfaces (CUI) soll eine intuitive Bedienbarkeit nach dem Vorbild natürlichsprachlicher Dialoge ermöglicht werden. Die Bereitstellung der Dienste erfolgt auf Basis eines User Interface Agent (UIA).
- *Basisagenten*:
  Entwicklung einer agentenbasierten Infrastruktur für System- und Benutzerdienste. Bereitstellung von typischen Funktionen von Anwendungsprogrammen aus dem Büro- und Verwaltungsbereich als anwendungsunabhängige Basisagenten. Bereitstellung von Werkzeugen zur Steuerung und Konfiguration der Agenten.
- *Mobile Basis und Assistenz*:
  Bereitstellung des mobilen Zugangs zu den MAP-Basisagenten, um sie zu jeder Zeit und an

jedem Ort nutzen zu können. Adaption der teilweise begrenzten Systemressourcen bzgl. der Darstellungsfähigkeiten, Verarbeitungsleistung, Speichermenge und Kommunikationsbandbreite.

6.8.2 Agentensystem

Der Kern des MAP Systems bildet sich aus den durch das Zusammenwirken zweier Agentenplattformen, zum einem der SeMoA Plattform und zum anderen der JADE Plattform [JADE]. Die beiden Plattformen sind so integriert, dass Sicherheitsaspekte, Mobilität der Agenten und eine gewisse Standardisierung durch die Einhaltung der FIPA Konformität in ausreichendem Maß erreicht werden kann.

Zur Kommunikation zwischen den Agenten der Plattform wird FIPA ACL eingesetzt. Das System ist in der Lage die Ausführung der Prozesse und die Haltung der Daten dynamisch zu verteilen. Auf diese Weise können die Softwareagenten selbständig im Netz agieren und migrieren, Informationen sammeln und Aufgaben ausführen. Dies alles ohne eine fortwährende Verbindung zum einem besonderen Host-Rechner oder zum Benutzer zu benötigen.

Der Arbeitsablauf gestaltet sich in etwa in folgender Weise: Der Benutzer spezifiziert eine Aufgabe mit einem Aufgabenziel, die er von einem Agenten erledigt haben möchte. Nach der Spezifikation der Aufgabe beginnt der beauftragte Agent die Bearbeitung und sucht die dafür notwendigen Ressourcen im Netzwerkverbund des MAP-Systems. Nachdem das Aufgabenziel erreicht wurde, migriert er zum System des Benutzers und übergibt ihm das Ergebnis. Ein solcher mobiler Agent[22] ist spezialisiert auf die Erledigung bestimmter Aufgabentypen (wie etwa das Auffinden einer Adresse) aber er ist nicht zwingend dazu fähig, die Präsentation der Interaktionsdaten auf beliebigem Endgerät vorzunehmen. Diese Aufgabe wird vielmehr durch einen speziellen UIA vorgenommen. Auf den UIA gibt es dabei zweierlei Sichtweisen: Aus der Sicht der Agenten im System ist der User Interface Agent eine Schnittstelle zum Benutzer. Dadurch spezifiziert und standardisiert er diese Schnittstelle zum Benutzer. Für den Benutzer bildet der UIA die Schnittstelle zu den Agenten.

6.9 Integration des User Interface Agenten in MAP

Die Aufgaben des UIA im MAP System sind vielfältig, da er in vielen Anwendungsfällen die einzige Möglichkeit des Benutzers darstellt, in das MAP System einzugreifen und es zu kontrollieren. Die Anforderungen an den UIA waren durch MAP wie folgt spezifiziert:

- Der UIA sollte neben graphischen WIMP basierten Oberflächen auch konversationale bzw. dialogorientierte Methoden beherrschen können.

[22] Ein solcher Agent kann auch als Workflow-Agent bezeichnet werden.

- Der UIA sollte verschiedene Endgeräte und Interaktionsmetaphern adaptieren können.
- Der UIA sollte mit multimodalen Interfacemethoden zurechtkommen.
- Der UIA sollte den Kontext der Interaktion berücksichtigen können.
- Es sollte eine anthropomorphe Variante zur Erprobung des konversationalen Ansatzes geben.

Damit ist der Katalog der Anforderungen, die das MAP System an das realisierte Konzept stellt, um die konversationalen Methoden und die anthropomorphe Darstellung reicher, als im Rahmen dieser Arbeit gefordert (vgl. Abschnitt 2.3).

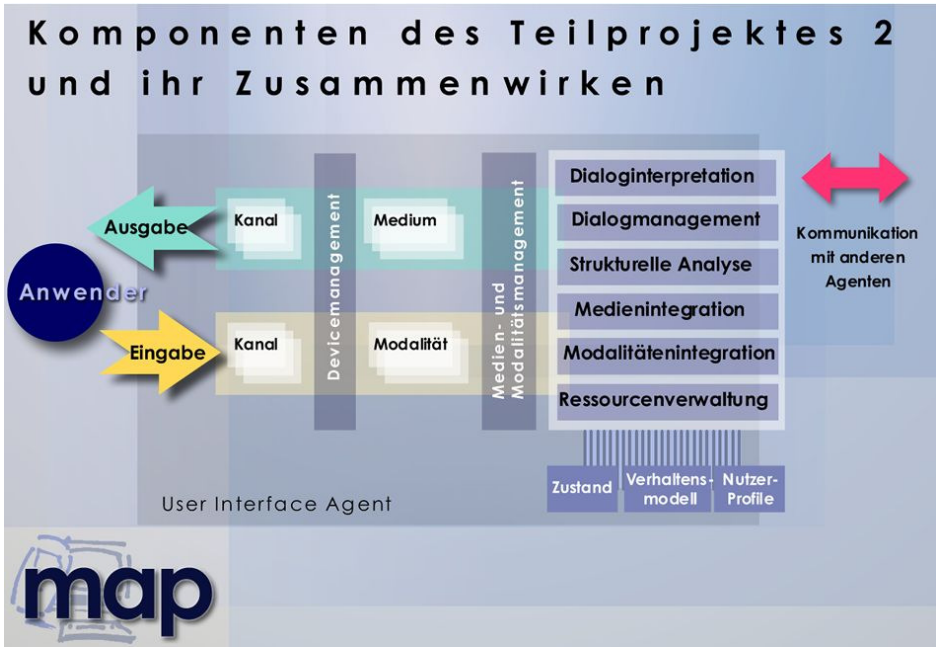

Abbildung 75: Die Komponenten des MAP – User Interface Agenten [KP01]

Die Abbildung 75 zeigt das Architekturbild des UIA. Aus der Beschaffenheit der Ein- und Ausgabekanäle folgen einerseits die Notwendigkeit zur Adaptionsfähigkeit und Abstraktion und andererseits eine gewisse Fähigkeit zur Steuerung einer multimodalen Schnittstelle. Die Ein- und Ausgabekanäle setzen sich aus allen in MAP eingesetzten Geräten und deren Schnittstellen zusammen. Diese Schnittstellen müssen durch die standardisierte Kommunikation auf der Agentenseite bedienbar sein, unabhängig von der gerade verwendeten Konfiguration. Jede in MAP eingesetzte Benutzungsoberfläche setzt sich aus einer Anzahl solcher adaptierten Kanäle zusammen. Diese müssen synergetisch vom UIA genutzt werden. Bei vordefinierten Benutzungsoberflächen, bei denen im Vorfeld jede Modalität bekannt ist, können

Verknüpfungen bei bestimmten Ereignissen in der Interaktion zugewiesen werden[23]. In einem System, das vorher nicht bekannte Modalitäten benutzt, können Mechanismen aus dem Bereich der multimodalen Interaktionssysteme zur Verknüpfung verschiedener Modalitäten herangezogen werden [Bolt80].

Die höheren Funktionen des UIA befinden sich um rechten Teil von Abbildung 75, wo eine Interpretation der Beschreibungssprache für die Mensch-Maschine Dialoge stattfindet, die Synchronisation der Ein- und Ausgabekanäle erfolgt und die Interaktion den Agenten über die Agentenkommunikation zugänglich gemacht wird.

6.10 Rendermodule

In diesem Abschnitt sind die in MAP verwirklichten Rendermodule für die Darstellung der Benutzungsoberfläche auf den verschiedenen Modalitäten bzw. Geräten dargestellt.

6.10.1 Konsolen-Rendermodul

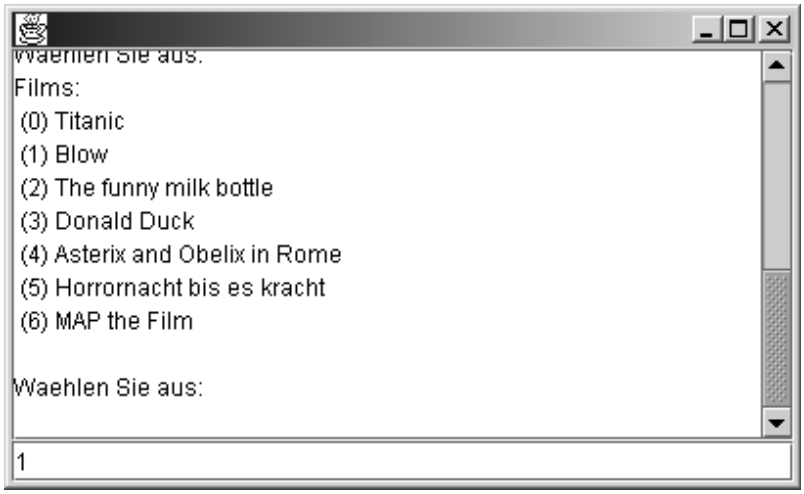

Abbildung 76: Ein Beispieldialog wird auf dem Konsolen Rendermodul präsentiert

Das Konsolen Rendermodul dient der Bereitstellung von Interaktion auf einer einfachen textbasierten Benutzungsoberfläche. Zum einen kann diese Präsentationsform sehr vorteilhaft

[23] Bei einem fensterbasierten System sind z.B. bestimmte Tasten übergreifend für bestimmte Funktionen reserviert. „F1“ für Hilfe oder der „rechte Mausklick“ führt zur Anzeige eines Kontextmenüs, etc..

sein, wenn die Fähigkeiten des Endgeräts auf die Darstellung von Text begrenzt sind. Zum anderen kann ein solches Modul auch als Vorstufe zu einer sequentiellen Darstellung eingesetzt werden, wenn die Ausgabe um eine Wandlung von Text in Sprache ergänzt würde.

Die Präsentation des Dialogs erfolgt menübasiert. Der UIA zeigt dem Benutzer, welche Aktionen er zu einem Zeitpunkt ausführen kann und erwartet dementsprechende Eingaben. Die Darstellung der Texte erfolgt in einem mit dem Java GUI Toolkit erzeugten Fenster. Die Eingaben werden über die Tastatur und zunächst in einem gesonderten Textfeld am unteren Rand des Fensters entgegen genommen. Ein Beispieldialog der mit diesem Rendermodul dargestellt wurde ist in Abbildung 76 gezeigt.

6.10.2 Java-WIMP-Rendermodul

Dieses Modul benutzt die bei Desktop-PCs heute üblichen Methoden (WIMP) der grafischen Interaktion unter Benutzung der Java-swing-Klassen. Für die Darstellung der Datentypen der Dialogbeschreibung werden folgenden Java-Klassen benutzt:

Dialogelement	**Java-Klasse**
DialogOpening	`javax.swing.Jpanel`
DialogMove	`javax.swing.Jpanel`
DialogEnd	`javax.swing.Jpanel`
Bool	`javax.swing.JcheckBox`
Integer	`javax.swing.JtextField`
Float	`javax.swing.JtextField`
Character	`javax.swing.JtextField`
String	`javax.swing.JtextField`
StringList	`javax.swing.JradioButton` bis 5 Elemente
	`javax.swing.JcomboBox` bei mehr als 5 Elementen

Tabelle 12: Die Abbildung der abstrakten Datentypen auf konkrete javax-swing-Objekte

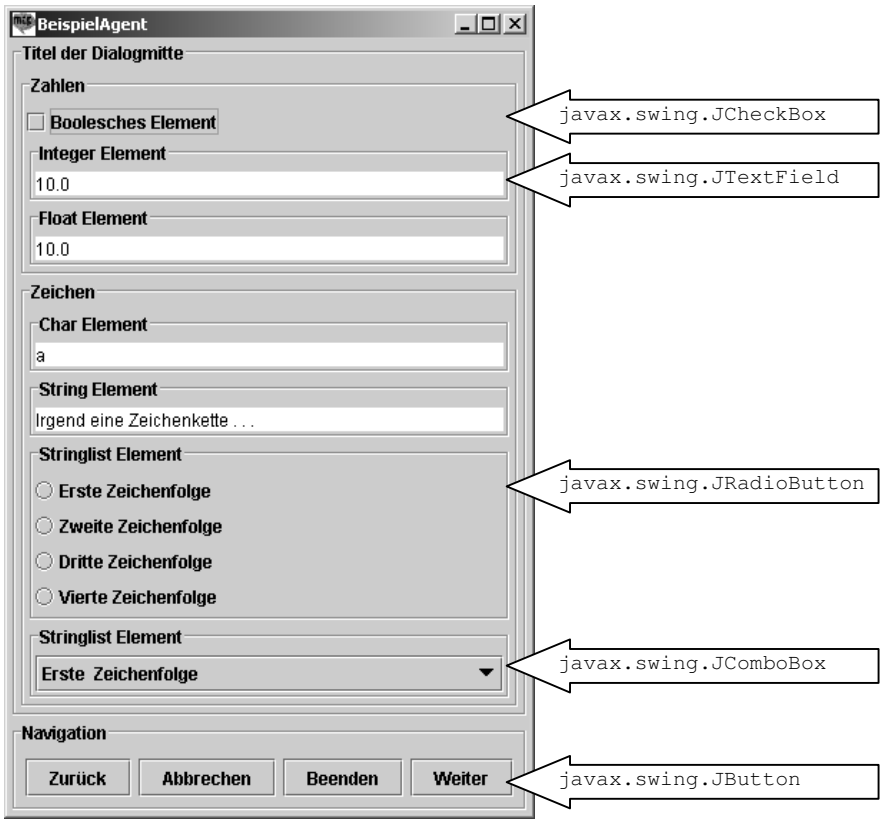

Abbildung 77: Die Elemente aus Tabelle 12 in einem Testdialog dargestellt

In obiger Abbildung 77 kann recht gut das Layoutmanagement des Java-GUI Rendermoduls erkannt werden. Der dargestellte Dialog ist hierarchisch organisiert eine Struktur gemäß Abbildung 78.

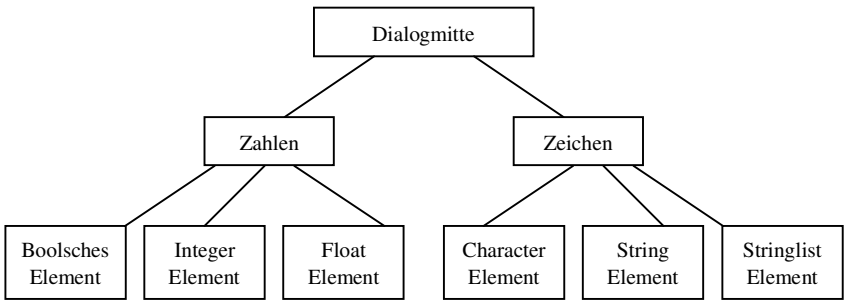

Abbildung 78: Die hierarchische Struktur des Beispieldialogs

Eine Besonderheit des GUI-Rendermoduls ist die unterschiedliche Darstellung des Stringlist Elements abhängig von der Anzahl der Optionen, zwischen denen der Benutzer wählen kann. Stehen mehr als fünf Optionen zur Auswahl, wird ein Platz sparendes `ComboBox`-Element angezeigt. Sind es hingegen weniger Optionen, werden sie alle gleichzeitig, unter Verwendung von `JradioButton`-Elementen angezeigt.

Das Rendermodul traversiert nach einem die Elemente, die durch den Parser aus der Dialogbeschreibung angefertigt wurde und ordnet sie der dabei auftretenden Reihenfolge nach untereinander an. Diese Konfiguration ist z.B. für hochkantige Anzeigeflächen ausgelegt worden.

Am unteren Rand der Benutzungsoberfläche des Beispieldialogs in Abbildung 77 befinden sich Bedienelemente zur Dialogkontrolle bzw. Navigation. Diese Elemente erlauben den Wechsel zurück zum Dialoganfang

6.10.3 HTML+ Rendermodul

Das HTML+-Rendermodul soll die mobile Interaktion auch über Internet-Browser ermöglichen. Der Vorteil besteht in der Verwendbarkeit ohne die Notwendigkeit zur Installation einer speziellen Softwareumgebung auf dem Endgerät. Dies bedeutet i. A. den Gebrauch von HTML-Formularen oder dazu ähnlichen Lösungen wie etwa Xforms (vgl. Abschnitt 4.2.7.1) oder XUL (vgl. Abschnitt 4.2.4). Für das MAP Projekt stand eine eben solche, über die Möglichkeiten von HTML-Formularen hinausgehende Lösung, zur Verfügung. HTML+ stellt eine Erweiterung des HTML-Standards dar und geht auf die Initiative von *Alcatel SEL AG Research and Innovation* zurück [RSWH+01].

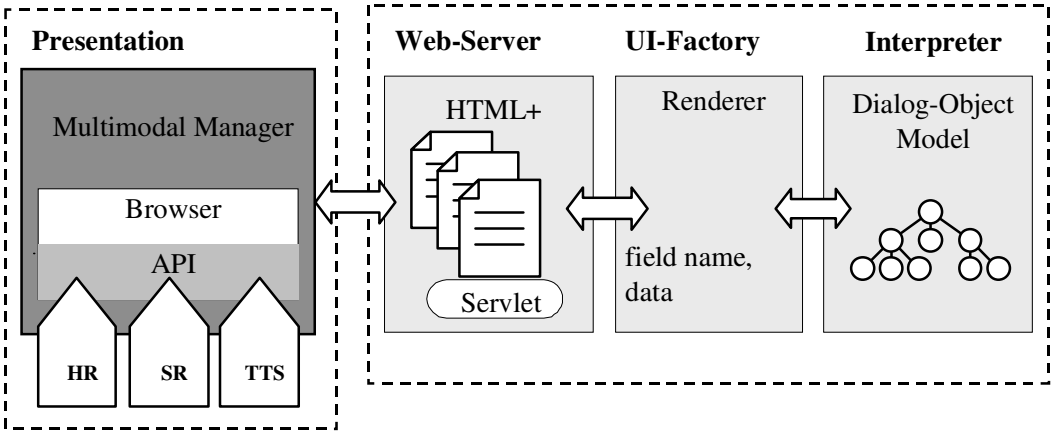

Abbildung 79: Die Komponenten des HTM+ Rendermoduls nach [BB03]

Im Gegensatz zu reinem HTML-Code unterstützt HTML+ durch Zugabe spezieller Tags multimodale Interaktionsmechanismen. Diese zusätzlichen Tags bleiben durch den Browser unberücksichtigt, wenn sie von ihm nicht unterstützt werden. So können HTML+ Seiten auch mit standardmäßig installierten Browsern in Grenzen angesehen werden. Der in MAP eingesetzte HTML+ Browser unterstützt außer den Standardfunktionen des HTML-Codes auch die Sprachausgabe und die Stifteingabe, die im MAP-Projekt mit einer Handschriftenerkennung verbunden ist.

Die Aufgabe des HTML+ Rendermoduls bestand darin das Objektmodell der Dialoge in entsprechende HTML+ Seiten zu überführen. Im Ergebnis wurde jeder Dialogschritt der Dialogbeschreibung in eine eigene HTML+ Seite übersetzt und auf einem WWW-Server zur Verfügung gestellt.

6.10.3.1 Generierung des HTML+Codes

Nach dem Parsen des der Dialogbeschreibung wird das Dialog-Objekt-Model aufgebaut. Danach werden alle Konstruktoren der Dialogelemente aufgerufen. Da die HTML+ Seiten, sowie die zugehörigen Sprachverarbeitungs-Dateien (Bnf-Dateien) auf einmal erzeugt werden, geschieht dies innerhalb der Konstruktoren. Somit ist also jeder Knoten für die Erzeugung seines eigenen HTML+ Codes und der dazugehörigen Bnf-Files zuständig. Informationen, die mehrere Knoten betreffen, wie zum Beispiel die Titel einzelner Dialogelemente, die auf einer gemeinsamen HTML+ Seite dargestellt werden sollen, werden während des Aufbaus des Objekt Modells gesammelt, es erfolgt aber kein zweiter Durchlauf der Baumstruktur. Genauso wird auch der erzeugte HTML+ Code nach oben im Baum weitergereicht und mit einem Schreibvorgang in eine HTML-Datei geschrieben.

Insgesamt werden drei HTML+ Seiten erzeugt, eine für den Dialoganfang, eine für den Mittelteil (enthält alle Knoten der Dialogmitte, die so genannten *Moves*, präsentiert mit einem Formular) und ein Dialogende. Der Mittelteil enthält dabei die Daten, die der Benutzer an den Dialog-Interpreter mittels des Servlets zurücksenden kann. Diese drei Seiten werden zusammen mit den Bnf-Files direkt in das Verzeichnis eines Web-Servers geschrieben.

6.10.3.2 Rückkanal-Servlet

Um die Interaktion zu ermöglichen, müssen auf Daten vom Browser zum HTML+ Server gesendet werden. Dies geschieht über das Rückkanal Servlet. Der WWW-Server der HTML+ Seiten hält das Rückkanal Servlet bereit, das die vom Benutzer in das Formular eingegebenen Daten verarbeitet. Das Servlet schreibt die erhaltenen Daten auf ein Socket[24], das mit dem Rendermodul verbunden ist.

Die Verarbeitung der Daten geschieht folgendermaßen: Die Daten des Formulars werden in eine Datenstruktur mit sog. Key-Value-Paaren gespeichert. Anschließend wird das Dialog-Objekt-Modell traversiert und die `interpret()`-Methode eines jeden Dialogelements aufgerufen. Nun

[24] Mit *Socket* wird ein Kommunikationskanal in einem IP-Netzwerk bezeichnet

vergleicht jedes Dialogelement, ob sich der eigene Name in der Datenstruktur befindet. Ist dies der Fall, wird der Durchlauf eines Event Request Protokolls durch das Dialogelement ausgelöst.

Die etwas umständliche Herangehensweise ist notwendig, da der HTML-Server das Dialog-Objekt-Modell vom HTML+-Browser trennt. Der Browser kann also nicht direkt mit einem Dialogelement im Interpreter kommunizieren, das Eventmodell des Interpreters benötigt jedoch nicht nur den Namen eines Event sendenden Objekts, sondern das Objekt selbst. Die technische Lösung des Problems bestand darin, den eindeutigen Namen des Dialogelements in die HTML+ Seiten zu schreiben und später das entsprechende Objekt zu aktivieren.

6.10.3.3 Bedienung des Multimodalen Browsers

Folgende Tabelle gibt die Unterstützung des Browsers für bestimmte Modalitäten bzw. Kanäle wieder[25]:

Sprache	Eingabe per Mikrofon,
	Spracherkennung für deutsches Vokabular,
	Wortschatz mit HTML+ Seite verknüpft
	Ausgabe per Lautsprecher (mono: intern oder extern),
	Text-To-Speech TTS, weibliche Stimme (deutsch)
Stift	Eingabe am Touch Display,
	Berührung der entsprechenden Navigationsflächen, oder Schreiben auf ausgewiesenes Feld im Browser und online Handschrifterkennung
Finger	Eingabe am Touch Display,
	Berührung der entsprechenden Navigationsflächen
Grafik	Ausgabe im Grafik-Feld auf Display
	Farbe 1024 x 768 Bildpunkte, intern oder extern
Bild	Assistent (Avatar) Video Clip im Grafik-Feld, männliche Stimme (deutsch)

Tabelle 13: Die vom HTML+-Browser unterstützten Modalitäten [BLB01]

Abbildung 80 zeigt die Benutzungsoberfläche beim Login Vorgang des multimodalen HTML+ Browsers. Durch den Zugriff über das Netzwerk wird eine Benutzeridentifikation erforderlich. Diese muss noch vor der eigentlichen Interaktion erfolgen.

[25] Touch Funktionen erfordern einen entsprechenden Bildschirm im mobilen Endgerät

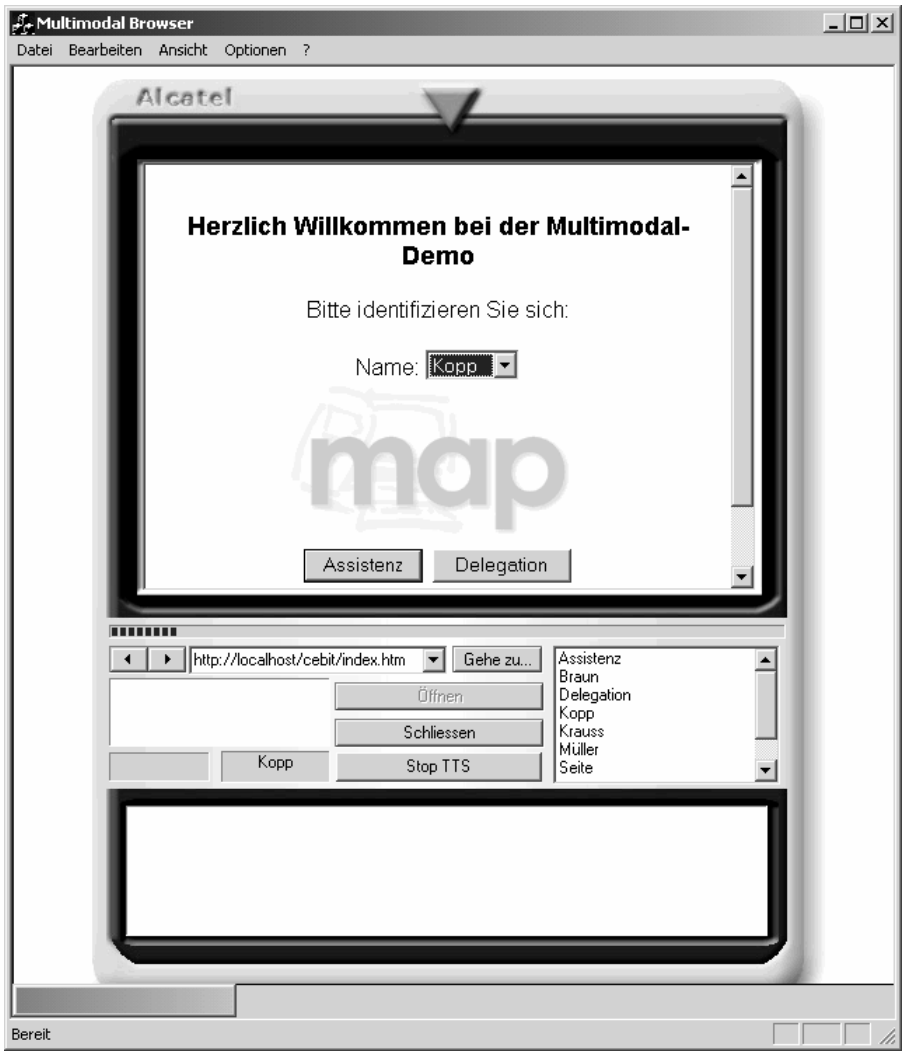

Abbildung 80: Die Benutzungsoberfläche des HTML+ Browsers [BLB01]

Die meisten Informationen können vom Benutzer über den visuellen Kanal (grafisches Ausgabefeld) aufgenommen werden. Die anderen Modalitäten bilden eine Ergänzung dazu und erweitern die Interaktivität. Alle Modalitäten sind gleichzeitig und ohne Umschaltung verfügbar.

Von der Startseite ausgehend sind 2 Betriebsweisen wählbar:

1. Assistenz Sequentielle Navigation mit multimodaler Unterstützung
2. Delegation Direkte Anweisungen mit sprachlicher Unterstützung

6.10.4 Avatar-Rendermodul

Das Avatar Rendermodul wurde entwickelt, um Assistenz- und Delegationsfunktionen durch eine anthropomorphe Darstellung mit einer adäquaten Metapher zu versehen. Der *Avatar* ist eine visualisierte Figur mit Fähigkeiten zur Darstellung menschlicher Mimik und Gestik. An den Avatar gekoppelt ist je ein Modul zur Spracherkennung und eines zur Sprachsynthese. Die genannten Modalitäten Avatardarstellung, Sprachein- und –ausgabe werden ihrerseits von einer *Conversation Engine* gesteuert [Brau02]. Diese Form der Interaktion zu erproben war Bestandteil des MAP-Projekts. Das Avatar Rendermodul stellt somit einen Adapter für die Conversation Engine dar.

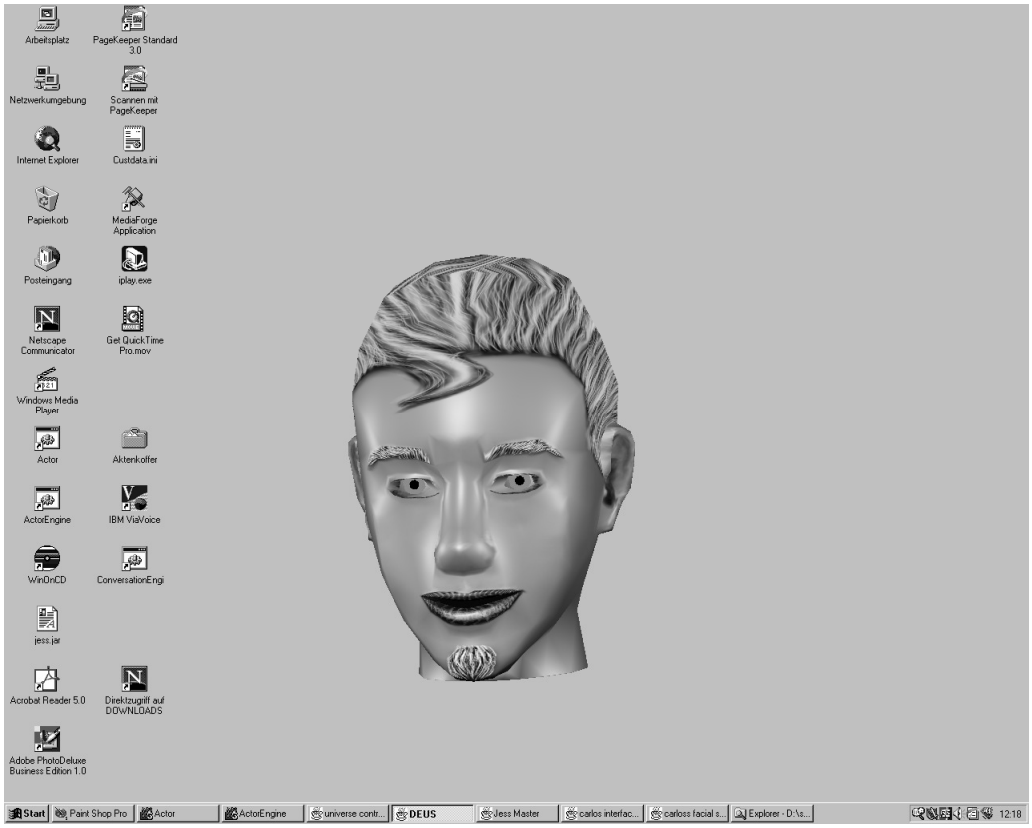

Abbildung 81: Die Ausgabe des Avatar Rendermoduls auf dem Bildschirm

6.10.4.1 Funktionsweise der Conversation Engine

Um die Adaption der Conversation Engine (CE) besser verstehen zu können soll ihre Funktionsweise in diesem Abschnitt skizziert werden.

Die CE basiert ihrerseits auf einer Story Engine. Diese Story Engine ist in Grenzen in der Lage sich mit dem Benutzer über Geschichten (Story) zu unterhalten. In der MAP Anbindung der CE sind diese Storys mit Gesprächsthemen assoziiert. Die CE arbeitet auf Basis von Verhaltensanweisungen, die sie in Form von Animationsaufrufen an die *Avatar Engine* weitergibt, die für die Animation des Avatars zuständig ist. Die Animationsaufrufe werden mit den sonstigen Animationen des Avatars vereinigt und durch die Avatar Engine abgespielt. Folgende Animationsabläufe können getätigt werden:

- Eröffnung,
- Reden,
- Zuhören,
- Rederecht abgeben,
- Rederecht holen,
- Abschließen,
- Diskurs starten,
- Diskurswechsel und
- Diskurs beenden.

Der Ablauf einer Verhaltenssequenz wird durch ein Verhaltenselement ausgelöst, welches von der CE übergeben wird. Wenn ein Agent etwas mitzuteilen hat, so gibt der dies an die CE weiter, diese teilt das in einem zusätzlich übergebenen Inhaltselement mit.

Die Verhaltensdauer eines Ablaufs ist jeweils an entsprechende zwischenmenschliche Situationen angelehnt. Neben reinen Verhaltens- und Inhaltsangaben werden folgende Faktoren zusätzlich berücksichtigt:

- Jedem Verhalten können Intensitätsfaktoren aus der Menge {FORCIEREN, NEUTRAL, DÄMPFEN} zugewiesen werden. Der Intensitätsparameter wird bei der Animation berücksichtigt.
- Die Inhaltselemente besitzen die Parameter *Dringlichkeit* und *Wichtigkeit*. Diese Parameter wirken ebenfalls auf die Animation.

Im Gegensatz zum UIA, der für einen Dialog immer nur zwei Gesprächspartner zulässt, beherrscht die CE prinzipiell mehrere Teilnehmer.

6.10.4.2 Anbindung

Die Kopplung der Conversation Engine mit dem UIA erfolgt über eine XML-basierte Schnittstelle. D.h. die Verhaltensanweisungen an die CE werden in Form kleiner XML-Dokumente gesendet und von ihr verarbeitet [Brau01].

Das Avatar Rendermodul des UIA setzt das Dialog Objekt Modell einer UIML-Dialogbeschreibung in eine Folge XML-kodierter Anweisungen um. Die CE bezieht sich immer

auf sog. *Diskurse*. Diese Diskurse sind nicht auf zwei Gesprächspartner begrenzt. Ein Diskurs wird mittels des ‚Diskurs'-Elements eröffnet. Innerhalb eines Diskurses wird eine Anzahl Konversationsteilnehmer festgelegt. Folgendes Beispiel zeigt einen entsprechenden Ausschnitt:

```
<Diskurs name="$DiskursName$">
     <konversationsteilnehmer> $Name_1$ </konversationsteilnehmer>
          …
     <konversationsteilnehmer> $Name_n$ </konversationsteilnehmer>
</Diskurs>
```

Die Bezeichner des obigen Beispiels haben die folgenden Aufgaben:

- `name` bezeichnet den Diskurs eindeutig.
- `Konversationsteilnehmer` bezeichnet den Konversationsteilnehmer.

Informationsinhalte werden in Story-Tags verwaltet. Es können sequentielle und asynchrone Informationen definiert werden.

```
<Story name="$NameStory$" typ="$TypStory$">

     <inhalt>  $content_1$ </inhalt>
     …
     <inhalt>  $content_n$ </inhalt>

</Story>
```

Den Elementen kommt folgende Bedeutung zu:

- `Name`
  ist der eindeutiger Bezeichner der Story
- `Typ`
  ist ein Element aus der Menge {sequentiell, asynchron}
- `Inhalt`
  bezeichnet den Namen der Inhaltselemente

Ist der Typ der Story sequentiell, so gibt die Reihenfolge der Inhalte im Story-Tag auch die Reihenfolge der Inhaltspräsentation (zum Benutzer hin) an.

Inhalte selbst werden mittels des Inhalt-Elements angegeben.

```
<Inhalt   name="$NameInhalt$"
          typ="$TypInhalt$"
          bezug="$BezugInhalt$"
          diskurs="$DiskursName$"
          dringlichkeit="$DieDringlichkeit$"
          wichtigkeit="$DieWichtigkeit$">

     <content> $Hier steht der Inhalt…$   </content>

</Inhalt>
```

- `name`
  bezeichnet den Inhalt eindeutig. Der Name sollte mit dem Story-Element `<inhalt>` übereinstimmen.
- `typ`
  ist ein Element aus {FRAGE, ANTWORT, AUSDRUCK}
- `bezug`
  bezeichnet den Namen eines Inhaltselements, auf das sich der Inhalt bezieht (z.B. hat eine Antwort immer einen Bezug zu einer Frage).
- `content`
  Der zu vermittelnde Inhalt
- `diskurs`
  Name des entsprechenden Diskurses
- `dringlichkeit`
  ist ein Element aus {WENIG, MITTEL, VIEL}
- `wichtigkeit`
  ist ein Element aus {WENIG, MITTEL, VIEL}

Ein Inhaltselement kann auch direkt als Antwort definiert sein. Dann kommt das Element `wiederholungen_erwünscht` hinzu:

```
<Antwort  name="$NameInhalt$"
          typ="$TypInhalt$"
          bezug="$BezugInhalt$"
          diskurs="$DiskursName$"
          dringlichkeit="$DieDringlichkeit$"
          wichtigkeit="$DieWichtigkeit$"
          wiederholungen_erwünscht="TRUE">

     <content> $Der Inhalt steht hier…$ </content>

</Antwort>
```

`wiederholungen_erwünscht` ist ein Element aus {TRUE, FALSE}.

Ebenso kann ein Inhaltselement direkt als Frage definiert sein:

```
<Frage    name="$NameInhalt$"
          typ="$TypInhalt$"
          bezug="$BezugInhalt$"
          diskurs="$DiskursName$"
          dringlichkeit="$DieDringlichkeit$"
          wichtigkeit="$DieWichtigkeit$"
          wiederholungen="3">

     <content> $Der Content steht hier…$ </content>

</Frage>
```

Die Conversation Engine kann Rückfragen an den Ersteller eines Diskurses senden. Diese werden mit dem Anfrage-Element definiert:

```
<Anfrage bezug="$NameInhaltselement$">
```

`bezug` bezeichnet einen Inhalt eindeutig.

6.10.4.3 Zusammenspiel mit dem User Interface Agenten

Die CE bietet eine Teilmenge der Grundoperationen, die für eine Dialogausführung durch den UIA bedient werden. Da die CE den sprachlichen Kanal zur Eingabe benutzt, kann sie nur bestimmte einfache Datentypen entgegennehmen. Dies stellt die Verwendung insofern vor kein Problem, als dass das Avatar-Rendermodul für die Interaktion begleitende Zwecken entwickelt wurde. In dieser Funktion geschehen die Eingaben über mindestens ein weiteres, gleichzeitig arbeitendes Rendermodul, das auch komplexere Eingaben versteht.

Die Abbildung der vom UIA verstandenen Elemente der Dialogbeschreibungssprache findet sich in Tabelle 14.

XML des UIA	XML-Element der CE
Dialog	Diskurs
DialogOpen	Story
DialogMove	Story
DialogClose	Story
BoolElement	-
IntegerElement	-
FloatElement	-
StringElement	-
StringlistElement	-

Tabelle 14: Abbildung der Sprachelemente des UIA auf Sprachelemente der CE

Wie aus der Tabelle geschlossen werden kann, liefert die CE dem UIA tatsächlich keine Eingaben des Benutzers, da keine Eingabeelemente angebunden wurden. Die Steuerung geschieht rein auf der Basis von Navigationsereignissen, die zwischen den Rendermodulen kommuniziert werden.

6.11 Beurteilung des User Interface Agenten durch Benutzer

Die Machbarkeit des UIA wurde anhand der Realisierung gezeigt. Die Benutzbarkeit wurde anhand einer psychologischen Beurteilung der Akzeptanz und Akzeptabilität mit den in MAP eingesetzten Systemkomponenten[26] geprüft. Dies erfolgte im Hinblick auf Aufgabendelegation und Assistenz durch den Benutzer mittels des MAP UIA [FT02]. Eine wichtige Voraussetzung für die Akzeptanz eines jeden Produktes ist die Möglichkeit des Benutzers, dieses Produkt tatsächlich zu gebrauchen. Die Gebrauchstauglichkeit eines Produktes impliziert dabei sowohl die Nützlichkeit als auch die Bedienbarkeit des Produktes. Um einen nachvollziehbaren Wert für die Gebrauchstauglichkeit eines Produktes zu erhalten, wurden folgende Punkte untersucht:

- Effektivität: Bis zu welchem Grad werden die Arbeitsziele erreicht?
- Effizienz: In welcher Relation stehen Arbeitsaufwand und erreichte Ziele?
- Akzeptanz: Wie zufrieden ist der Benutzer des Produktes?

Als prototypisches Szenario wurde das Bauwesen ausgewählt [Han00]. Arbeiten wie Projektsteuerung, Bauüberwachung bzw. in Architekturbüros die komplette Überwachung der Bauphasen Planung, Bau und Nutzung sind mit einem erheblichen Wechsel zwischen Phasen der Bürotätigkeit und Tätigkeiten vor Ort (auf der Baustelle) und dadurch mit einem hohen Koordinationsaufwand verbunden. Die Hilfestellung, die in MAP mittels des UIA erreicht wurde, konnte durch den hohen Anteil an delegativen Aufgaben effizient überprüft werden.

Die generischen Applikation MAP ist in verschiedenen Szenarien eingesetzt und evaluiert worden. Die Evaluation der Applikationen erfolgte im Hinblick auf die Anwendungsvorgaben und Anwendungsziele – aus diesem Grund werden die einzelnen Anwendungen im Kontext ihrer Anwendungsszenarien beschrieben und die entsprechende Evaluation vor dem Hintergrund der jeweiligen Anwendungsszenarien diskutiert.

Der MAP User Interface Agent wurde anhand der in Kapitel 2 vorgestellten Kriterien nach ISO 9241-10 untersucht. Die Aufgaben des UIA ist die Vermittlung zwischen dem Anwender und den Basisagenten des MAP Systems – und dies unabhängig von den Diensten, welche die Basisagenten im anbieten können.

[26] Diese Beurteilung wurde durch das Siemens Usability Labor [Sh01] durchgeführt

Aufgaben-angemessenheit	Der Dialog mit dem Benutzer unterstützt denselben angemessen, um die Aufgaben des Benutzers zu einer effizienten und effektiven Erledigung zu bringen. Die Angemessenheit erreicht dabei annähernd die im Projektziel geforderte delegative Güte und Assistenz.
Selbstbeschreibungs-fähigkeit	Durch den Einsatz der konversationalen Metapher ist der Dialog in hohem Maße selbstbeschreibungsfähig. Jeder einzelne Dialogschritt wird dem Benutzer durch konversationale Rückmeldung visualisiert.
Steuerbarkeit	Durch die bewusste explizite Konversationsmodellierung wird dem Anwender jederzeit die Möglichkeit gegeben, den Dialog zu beeinflussen. Dies wird durch die Offerte eines Sprecherwechsels unterstützt.
Erwartungs-konformität	Die Lernförderlichkeit wird im MAP System durch den Einsatz eines speziellen Hilfe-Agenten erreicht.
Fehlerrobustheit	Durch den Einsatz von multimodalen Techniken kann eine Fehlerrobustheit gegenüber Fehleingaben des Benutzers erreicht werden, die deutlich höher ist als die Robustheit gängiger GUI.
Individualisier-barkeit	Die Individualisierbarkeit des UIA ist sowohl in Bezug auf Arbeitsabläufe sowie Nutzerpräferenzen als auch in Bezug auf das individuelle Aussehen des konversationalen Charakters gegeben. Sie erfolgt über eine freie Wahl des Benutzers bezüglich der Animationsgrundlage des Charakters ebenso wie über Userprofiling und Benutzer-spezifische Definition von Agentenaufgaben.
Lernförderlichkeit	Die Lernförderlichkeit wird im MAP System durch den Einsatz eines speziellen Hilfe-Agenten erreicht.

Tabelle 15: Beurteilung des MAP-User Interface Agenten nach ISO 9241-10 [SH01]

Das Siemens Usability Labor [SH01] kam zum Schluss, dass der MAP User Interface Agent aus ergonomischer Sicht sehr sinnvoll ist. Die Ergebnisse der Untersuchung sind in Tabelle 15 zusammengefasst.

6.12 Zusammenfassung

Gegenstand dieses Kapitels ist der Nachweis der Realisierbarkeit des in Abschnitt 5 vorgestellten Konzepts für den UIA und seiner Architektur, das die Anforderungen aus Abschnitt 2 erfüllt. Die vorgenommene Realisierung verfolgt konsequent die Vorgaben des Konzepts und zeigt, dass die Kernaufgabe des UIA, nämlich die abstrakte Darstellung der Interaktionsdaten und die Anpassung an verschiedene Endgeräte mit dem System möglich ist.

Die Adaption erfolgt durch so genannte Rendermodule, welche die Präsentationslogik realisieren. Eine Besonderheit des Systems stellt die Möglichkeit dar, mehrere Rendermodule

zur selben Zeit auf einem Endgerät betreiben zu können. Dies wurde erreicht durch die Einbeziehung multimodaler Techniken und durch strukturelle Maßnahmen. So ist jeder Dialog einschließlich seiner Elemente einmal abstrakt und ein- bis mehrfach konkret durch Rendermodule repräsentiert. Die Dialogelemente sind hierbei hochgradig miteinander vernetzt, was die Synchronisation der Ein- und Ausgaben auf mehreren Ebenen ermöglicht.

Die Neuerstellung von Rendermodulen ist Teil der Anwendung des Systemansatzes. Hierfür wurde ein Framework für Rendermodule erstellt, das in MAP der Ausgangspunkt für die Programmierung mehrerer Rendermodule war, mit denen sich folgende Arten von Benutzungsoberflächen unterstützen lassen:

- WIMP basierte Benutzungsoberflächen,
- textbasierte und menügeführte Benutzungsoberflächen,
- WWW-basierte Interaktion über einen multimodalen HTML+-kompatiblen Web-Browser und
- Conversationale Interaktion, die Assistenz gebend eingesetzt werden kann.

Zusammenfassend kann festgestellt werden, dass die Integration des UIA in das MAP Projekt durchgeführt werden konnte. Es hat sich gezeigt, dass das Konzept des UIA derart flexibel gestaltet ist, das die notwendigen Anpassungen ohne Probleme erreicht werden konnten.

7 Vergleich mit existierenden Systemen

Nachdem im vorangegangenen Kapitel die Realisierbarkeit des Systems nachgewiesen wurde und auf eine erfolgreiche Prüfung des Prototypen nach ergonomischen Gesichtspunkten im MAP-Kontext verwiesen wurde, soll die gefundene Lösung konkurrierenden Systemen gegenüber gestellt werden. Deshalb widmet sich dieses Kapitel dem Vergleich mit diesen Systemen, die teils kommerziell verfügbar sind, bzw. prototypischen Charakter aufweisen und die Gegenstand von Forschungsvorhaben sind. Um einen Vergleich durchführen zu können werden zunächst einige Vorüberlegungen angestellt bezüglich der Vorgehensweise und der Vergleichbarkeit der Systeme. Anschließend werden die Systeme skizziert und nach funktionalen Gesichtspunkten bewertet. Die Bewertung wird anhand von in Kapitel 2 aufgestellten Bewertungskriterien vorgenommen.

7.1 Bewertungskriterien

Um die gefundene Lösung mit den existierenden Systemen vergleichen zu können, werden Bewertungskriterien auf die betrachteten Systeme angewendet. In Kapitel 2 wurden funktionale Anforderungen an die Systemarchitektur identifiziert, die sich aus *anwendungsorientierten Anforderungen*, *benutzerorientierten Anforderungen*, *entwicklerorientierten Anforderungen* und *allgemeinen Anforderungen* zusammensetzen. Wie im Rahmen des MAP-Projekts zweifelsfrei festgestellt wurde, sind die funktionalen Anforderungen durch den UIA, der die Konzepte dieser Arbeit implementiert, erfüllt.

Für den Vergleich zwischen den Systemen kann eine Auswahl an Kriterien getroffen werden, die näher konkretisiert werden. Der Kriterienkatalog aus Kapitel 2 war noch auf die Schaffung eines allgemeinen Konzepts und die Bereitstellung einer Architektur ausgerichtet. In Kenntnis der in Kapitel 4 betrachteten existierenden technischen Verfahren und Systeme, die durch den Stand der Technik gegeben sind, kann eine Auswahl an Kriterien getroffen werden, die einen direkten Vergleich zulassen.

Das realisierte System ermöglicht den Einsatz in einer heterogenen Umgebung, die sich aus verschiedenen Betriebssystemen zusammensetzen kann. Die Betriebssysteme sind auf bestimmte Anwendungsfelder ausgerichtet. Zwar konkurrieren die Betriebsysteme innerhalb eines Anwendungsfeldes miteinander, die Rivalität reicht jedoch nicht über das Anwendungsfeld hinaus. Bei Mobiletelefonen wird meist *Symbian*, bei PDAs meist *PalmOS*, *PocketPC* oder *Windows Mobile 2003* eingesetzt. Bei Desktop- und Laptop-Systemen werden aktuell *WindowsXP* von Microsoft, *MacOS* von Apple und das bekannte Open Source Betriebssystem

Linux eingesetzt. Die Einbeziehung möglichst vieler Plattformen ist deshalb ein wichtiges Kriterium für die Beurteilung der Systeme.

Die Integration von mobilen und stationären Endgeräten zu einer Plattform ist weiterhin durch die Heterogenität der Endgeräteschnittstellen gekennzeichnet. Bei grafischen Benutzungsoberflächen ist die verfügbare Auflösung von Bedeutung. Ein Vergleich der heute üblichen Anzeigeauflösungen in Abschnitt 5.7 belegt dies sehr deutlich. Die Systeme werden daraufhin verglichen, ob sie Verfahren zur *Adaption von kleinen Anzeigen* unterstützen und ob diese Verfahren die *Struktur* der Dialoge *erhalten*. Bei kleinen Anzeigen gewinnt auch die *Navigation* an Bedeutung, deshalb wird geprüft, ob die Systeme hier unterstützend wirken. Darüber hinaus wird betrachtet, ob auch *assistenzgebende Metaphern* in das System integrierbar sind.

Die Dynamik des Marktes für mobile EDV und Telekommunikation erfordert die Integrierbarkeit künftiger Interaktionstechniken. Bei den betrachteten Systemen wird abgeschätzt, ob es möglich ist das jeweilige System mit geringem Aufwand an bestimmte Situationen anzupassen und ob es möglich ist das System für gänzlich neue Interaktionsschnittstellen zu erweitern.

Die Betrachtung des Stands der Technik zeigte, dass im Wesentlichen Browser-basierte Systeme eingesetzt werden. Agentensysteme und Frameworks wie .NET erlauben auch die Ausführung von Aufgaben des funktionalen Kerns auf den Frontends. Die Systeme werden daraufhin bewertet, ob die Betriebsarten *Remote Evaluation (REV), Code On Demand (COD)* möglich sind oder ob Programmcode im Sinne der *mobilen Agenten* auf dem Frontend bearbeitet werden kann (vgl. Abschnitt 3.5.2).

Die Anforderungen aus Kapitel 2 nach der Unterstützung der Mobilität wird erfüllt, wenn das betrachtete System den *Wechsel des Endgeräts* während der Interaktion erlaubt. Um den Dialog fortführen zu können muss der *Dialogzustand* durch den UIA festgestellt und zum Zielsystem übertragen werden. Der UIA auf dem Zielsystem muss ermitteln, welche Teile des Dialogs auf Basis der *Endgerätefähigkeiten* darstellbar sind und geeignet reagieren können.

In folgender Tabelle 16 sind die Kriterien noch einmal zusammengefasst.

Kriterien	**User Interface Agent**
Plattformen (stationär)	
WindowsXP	JAVA
MacOS	JAVA
Linux/UNIX	JAVA
Plattformen (mobil)	
PalmOS	JAVA/MIDP2.0
Windows Mobile	JAVA/MIDP2.0
Symbian	JAVA/MIDP2.0
Adaptionsverfahren	
Adaption kleiner Anzeigen	✓
Strukturerhaltung	✓
Verbessert Navigation	✓
Unterstützt Assistenz	✓
Wartbarkeit	
Einfache Anpassbarkeit	Auswahl der Rendermodule
Einfache Erweiterbarkeit	Durch neue Rendermodule
Betriebsarten	
Remote Evaluation (REV)	HTML-Rendermodul
Code On Demand (COD)	möglich
Mobile Agenten	JADE/JADE-LEAP
Offline	✓
Dialogmigration	
Dialogmigration auf anderes Gerät	✓
Erhaltung des Dialogzustands	✓
Berücksichtigung Gerätefähigkeiten	✓

Tabelle 16: Die Erfüllung der Kriterien durch den User Interface Agenten

7.2 MUSA

Das MUSA-System unterstützt, dank seiner Implementation auf Basis der JAVA virtuellen Maschine, alle in Tabelle 17 aufgelisteten Betriebssysteme. Durch die Entkoppelung von funktionalem Kern und Präsentationslogik ist die eigentliche Darstellung im Prinzip unabhängig vom Betriebssystem.

Es besteht in seinem Kern aus einem Event Handler Graphen, der eine interne Repräsentation des Dialogs ist. Die Event Handler (EH) stellen die Knoten des Graphen dar, die Beziehungen zwischen den EH sind die Kanten des Graphen. Jeder Knoten ist ein vollwertig ausführbares Dialogelement.

Das System realisiert ein *adaptives Clustering* Verfahren, welches die Anzahl der gleichzeitig dargestellten EH an den verfügbaren Platz auf der Anzeige variiert. Das adaptive Clustering nutzt die Struktur des EH-Graphen aus, um mehrerer Knoten zu einem Meta-Knoten zusammenfassen zu können. Dazu muss das Aufgabenmodell eine hierarchische Ordnung aufweisen. Das MUSA-System benutzt diese Ordnung, um die Dialogelemente für den Benutzer leicht nachvollziehbar darzustellen. Das Verfahren erlaubt die stufenweise Adaption der Anzeigefläche; ist wenig Fläche zur Präsentation vorhanden, kann die Darstellung auf ein Element, das einem Dialogschritt entspricht und einigen Navigationselementen, reduziert werden. Wenn die Darstellungsfläche vergrößert wird, können weitere Dialogelemente hinzugenommen werden, ohne die Notwendigkeit neue Navigationselemente anzubieten. Das Clustering wird durch entsprechende Navigationsfunktionen unterstützt. Assistenzgebende Bedienmetaphern werden durch MUSA nicht unterstützt.

Das MUSA-System benutzt sog. *JAVA-Beans* zur Darstellung der Dialogelemente auf der Präsentationsseite. Um eine gewünschte Präsentationsform einzurichten wird einfach ein passendes *JAVA-Bean* ausgewählt. Durch die Entwicklung eigener *JAVA-Beans* kann das MUSA System ohne weiteres auch an neue Technologien angepasst werden.

MUSA stellt adaptive Verfahren auf Basis einer Client-Server Architektur bereit. Daher sind lediglich die Betriebsarten *REV*, z.B. über einen HTML-Browser und über JAVA-Beans, möglich (vgl. Abschnitt 3.5.2). Der Betrieb mit Softwareagenten oder ein Offline Modus ist ebenso wenig vorgesehen, wie ein Wechsel des Endgeräts inmitten eines Dialogs.

Kriterien	**MUSA**
Plattformen (stationär)	
WindowsXP	✓
MacOS	✓
Linux/UNIX	✓
Plattformen (mobil)	
PalmOS	✓
Windows Mobile	✓
Symbian	✓
Adaptionsverfahren	
Adaption kleiner Anzeigen	Adaptives Clustering
Strukturerhaltung	Ja
Verbessert Navigation	Ja
Unterstützt Assistenz	Nicht vorgesehen
Wartbarkeit	
Einfache Anpassbarkeit	Auswählen eines *Bean*
Einfache Erweiterbarkeit	Entwickeln eines *Bean*
Betriebsarten	
Remote Evaluation (REV)	HTTP
Code On Demand (COD)	JAVA-Bean/Web Services
Mobile Agenten	Nein
Offline	Nein
Dialogmigration	
Dialogmigration auf anderes Gerät	Nein
Erhaltung des Dialogzustands	**--**
Berücksichtigung Gerätefähigkeiten	**--**

Tabelle 17: Die Erfüllung der Kriterien durch MUSA

7.3 Microsoft .NET

Das Microsoft .NET Framework unterstützt nur die Firmeneigenen Betriebsysteme. Dies sind die für Desktop- bzw. Laptop-PC ausgelegten Systeme *Windows2000*, *WindowsXP*, etc. Für mobile Endgeräte stehen entsprechend die mobilen Betriebssysteme *Windows Mobile*, *PocketPC* und *WindowsCE*.

Die Verfahren *Pagination* und *Chunking* stehen als Methoden zur Anpassung der Interaktion auf kleinere Bildschirme zur Verfügung. Die Dialogstruktur geht dabei jedoch verloren, weil bei dem Pagination-Verfahren die Elemente eines Dialogs in eine Sequenz kleiner Dialoge mit einem einzelnen Bedienelement transformiert werden. Die sich ergebende lineare Liste aus Dialogen kann lediglich durch vor- und zurück-Kommandos Navigiert werden. Das Chunking-Verfahren bezieht sich auf einen bestimmten Typus von Dialogelementen, nämlich das sog. Listenelement. Ein Listenelement bietet dem Benutzer eine Anzahl von Einträgen zur Auswahl an. Wenn die Einträge im Listenelement eine bestimmte Anzahl überschreiten, werden die Einträge ebenfalls auf mehrere Dialoge verteilt, die wiederum durch *vor-* und *zurück-*Kommandos navigiert werden können. Offensichtlich bekommt die Navigation bei dieser Form der Adaption eine zusätzliche Bedeutung. Das .NET Framework bietet dem Benutzer jedoch keine zusätzlichen Navigationsmethoden zur Bewältigung des zusätzlichen Navigationsaufwands. Es ist zudem Anzumerken, dass die adaptiven Verfahren lediglich den Web-Formularen zur Verfügung stehen. Die Windows-Formulare unterstützen diese Form der Adaption nicht. Die Windows Formulare nehmen statt dessen eine einfache Abbildung der Dialogelemente auf die Gerätetypischen Elemente vor. Schon bei üblichen Dialogen kann bei einem kleinen Endgerät schnell die Grenze des ergonomisch sinnvollen erreicht werden.

Keine der beiden Betriebsarten von .NET, *Web-Formulare* und *Windows Formulare* ermöglichen dem Anwender die *Migration* eines laufenden Dialogs von einem auf ein anderes Endgerät.

Kriterien	**Microsoft .NET**
Plattformen (stationär)	
WindowsXP	✓
MacOS	Nein
Linux/UNIX	Nein
Plattformen (mobil)	
PalmOS	Nein
Windows Mobile	✓
Symbian	Nein
Adaptionsverfahren	
Adaption kleiner Anzeigen	Pagination/Chunking
Strukturerhaltung	Nein
Verbessert Navigation	Nein
Unterstützt Assistenz	Nein
Wartbarkeit	
Einfache Anpassbarkeit	Nein
Einfache Erweiterbarkeit	Nein
Betriebsarten	
Remote Evaluation (REV)	Web-Formulare
Code On Demand (COD)	Windows-Formulare
Mobile Agenten	Nein
Offline	Nein
Dialogmigration	
Dialogmigration auf anderes Gerät	Nein
Erhaltung des Dialogzustands	**--**
Berücksichtigung Gerätefähigkeiten	**--**

Tabelle 18: Erfüllung der Kriterien durch Microsoft .NET

7.4 EMBASSI

Das EMBASSI-System besteht in seinem Kern aus einem Server, der eine integrative Plattform bereitstellt. Die Serverapplikation ist ein in JAVA implementiertes Programm, das prinzipiell auf jeder JAVA-fähigen Plattform in Betrieb genommen werden kann. Hier besteht im Sinne der Vergleichskriterien theoretisch keine Beschränkung, denn alle Plattformen werden unterstützt.

Die in EMBASSI eingesetzten Agenten sind nicht mobil und deren Kooperation basiert auf einem TCP/IP Protokoll. Somit kann es nicht in einem Offline Modus ohne Verbindung zum Server betrieben werden. EMBASSI realisiert gewisse Ansätze zur adaptiven Nutzung von mobilen Endgeräten mit Agenten [RE03]. Ähnlich wie beim .NET Framework wird eine Transformation der Dialogstruktur in eine nacheinander zu bearbeitende Folge von Teildialogen vorgenommen. Die Struktur des Dialogs bleibt hierbei erhalten und dem Benutzer werden zusätzliche Navigationsmittel angeboten.

Die dynamische Konfigurierbarkeit des Systems zur Laufzeit belegt die Eignung eines multimodalen Ansatzes zur Erreichung adaptiver Interaktion. EMBASSI assoziiert die Modalitäten mit den Geräteschnittstellen seiner Endgeräte. Die ad hoc Erweiterung des Systems durch neue Geräte und Komponenten, bedeutet daher auch die dynamische Adaption der neuen Geräteschnittstellen. Die Anpassung ist deshalb einfach durch die Auswahl eines Geräts und eines Benutzerprofils möglich. Die Erweiterung hingegen wird durch die Entwicklung einer neuen Client Anwendung vorgenommen, die das in EMBASSI verwendete TCP/IP-Protokoll unterstützt. Der Aufwand hierfür ist nur schwer einzuschätzen.

Aufgrund der stationären Agenten ist eine Behandlung von Migration nicht vorgesehen. Es erübrigt sich daher die Notwendigkeit die Dialoge zu unterbrechen, um sie auf anderen Geräten fortzuführen. Die Frontends benötigen ohnehin den Kontakt zur Serveranwendung, um die multimodale Interaktion durchführen zu können.

Kriterien	**EMBASSI**
Plattformen (stationär)	
WindowsXP	✓
MacOS	✓
Linux/UNIX	✓
Plattformen (mobil)	
PalmOS	✓
Windows Mobile	✓
Symbian	✓
Adaptionsverfahren	
Adaption kleiner Anzeigen	Durch Sequentialisierung
Strukturerhaltung	Nein
Verbessert Navigation	✓
Unterstützt Assistenz	✓
Wartbarkeit	
Einfache Anpassbarkeit	Durch Benutzerprofil
Einfache Erweiterbarkeit	Entwicklung eines Clients
Betriebsarten	
Remote Evaluation (REV)	✓
Code On Demand (COD)	Nein
Mobile Agenten	Nein
Offline	Nein
Dialogmigration	
Dialogmigration auf anderes Gerät	✓
Erhaltung des Dialogzustands	Nein
Berücksichtigung Gerätefähigkeiten	✓

Tabelle 19: Erfüllung der Kriterien durch EMBASSI

7.5 SmartKom

Die in SmartKom verwendete MULTIPLATFORM basiert auf einer Open Source Plattform, die das *UNIX* basierte *Sun Solaris*, *GNU-LINUX* und *Microsoft Windows* unterstützt [HKMN+03]. Darüber hinaus ist auch das mobile Betriebsysteme *Windows Mobile* bzw. *PocketPC* integriert. Genauere Informationen, welche Betriebsysteme in MULTIPLATFORM integrierbar sind, finden sich nicht. Eine Portierung auf MacOs und PalmOS scheint aber ohne weiteres möglich zu sein.

Da SmartKom die naturnahe Mensch-Maschine Interaktion fokussiert ist das System auf eine sehr leistungsfähige Infrastruktur angewiesen. Das spiegelt sich auch in den 3 leistungsfähigen Server-Rechnern[27] wieder, aus denen das Demonstrationssystem besteht [Wahl03]. Die Verteilung der Rechenleistung wird durch die Implementation des *Blackboard-Modells* erreicht. Die Teilprozesse des SmartKom Systems kommunizieren über das Blackboard miteinander, was eine nahezu parallele Arbeitsweise der Teilprozesse ermöglicht. Die verhältnismäßig geringe Performanz der mobilen Endgeräte, die als Frontend eingesetzt sind, kann so durch den Netzzugang kompensiert werden. Die Bearbeitung der ressourcenaufwändigen Funktionen wird dann auf andere Systeme verlagert. Dies hat zur Folge, dass die Nutzung der Mechanismen der natürlichen Dialoge für die Adaption von Endgerät und Kontext nur in Verbindung mit der Infrastruktur in Frage kommt. *Offline* sind diese Leistungsstarken Funktionen etwa auf einem PDA nicht nutzbar.

Die Interaktion in SmartKom ist auf *Assistenz* fokussiert und benötigt für den Einsatz in einem bestimmten Anwendungsbereich eine umfangreiche Wissensbasis, etwa um die gesprochenen Texte verstehen zu können. Derzeit stehen Wissensbasen für verschiedene repräsentative Szenarien zur Verfügung, wie etwa ein Manager für *Hotelbuchungen* oder ein *virtueller Stadtführer*. Die Erweiterung der Wissensbasis ist mit einem relativ hohen Aufwand verbunden, weshalb die *Erweiterung* bzw. *Anpassung* an bestimmte Anwendungen aufwandsbedingt nicht immer sinnvoll sein wird.

SmartKom stellt dem Benutzer einen Assistenten (Smartakus) zur Verfügung, der auf dem Frontend in Form einer Figur sichtbar ist. Dadurch stehen Mimik und Gestik als zusätzliches Ausdruckmittel für die Interaktion zur Verfügung. Während die Visualisierung des Assistenten an das Endgerät angepasst wird werden Verfahren zur Adaption von grafischen Benutzungsoberflächen (*Sequentialisierung*) an neue Bildschirmauflösungen nicht berücksichtigt. Da das System eine Wissensbasis auswertet, offenbaren die Dialoge eine vom Menschen *nachvollziehbare Struktur*, die dem Benutzer auch bei der Navigation im Dialog hilft. Die Umsetzung als Mixed-Initiative Interface wirkt bei der *Navigation* hilfreich. Wird innerhalb des Dialogs ein Begriff genannt, der dem System bekannt ist, kann der Dialogverlauf geändert werden, um auf das neue Thema einzugehen.

SmartKom ist grundsätzlich zum *Gerätewechsel* fähig. Der *Dialogzustand* wird ebenfalls migriert. Die *Fähigkeiten des Endgerätes* werden beim Wechsel berücksichtigt.

---

[27] Es werden 3 Dual Xeon Prozessoren mit je 2.8 GHz genannt.

Kriterien	**SmartKom**
Plattformen (stationär)	
WindowsXP	✓
MacOS	möglich
Linux/UNIX	✓
Plattformen (mobil)	
PalmOS	möglich
Windows Mobile	✓
Symbian	möglich
Adaptionsverfahren	
Adaption kleiner Anzeigen	Nur für Assistenz
Strukturerhaltung	✓
Verbessert Navigation	✓
Unterstützt Assistenz	✓
Wartbarkeit	
Einfache Anpassbarkeit	Durch Benutzerprofil
Einfache Erweiterbarkeit	Hoher Aufwand
Betriebsarten	
Remote Evaluation (REV)	Durch Blackboard
Code On Demand (COD)	
Mobile Agenten	Nein
Offline	Nein
Dialogmigration	
Dialogmigration auf anderes Gerät	✓
Erhaltung des Dialogzustands	✓
Berücksichtigung Gerätefähigkeiten	✓

Tabelle 20: Erfüllung der Kriterien durch SmartKom

7.6 Vergleichende Gegenüberstellung

Die auf folgender Seite dargestellte Tabelle 21 fasst die in diesem Kapitel angestellten Betrachtungen noch einmal zusammen und gibt einen Überblick. Die Prüfung der Systeme ergab, dass die mit dem UIA aus MAP konkurrierenden Systeme manche Teilprobleme in bestimmten Anwendungsfällen besser lösen. Jedoch ist dies nach dem gegenwärtigen Stand der Technik problembehaftet. So sind etwa umfangreiche Verfahren zur Sprach- Gesichts- und Gestenerkennung zum einen nur in Verbindung mit einer leistungsfähigen Infrastruktur möglich. Diese Infrastruktur muss bei der Interaktion fortwährend zugreifbar sein, womit das Arbeiten außerhalb des Wirkungsbereichs eines Netzes nicht möglich ist. Zudem sind auch solch leistungsfähige Verfahren nicht in allen Situation angemessen zu verwenden. Die Nutzung von Sprache setzt entsprechend niedrige Geräuschpegel voraus, Gestik als Eingabemethode benötigt eine Kamera, welche auf die Hände gerichtet ist.

Insgesamt kann aufgrund der Anforderungen die Tragfähigkeit des Ansatzes anhand der Möglichkeiten des UIA nachgewiesen werden. Dies ist auf mehrere Faktoren zurückführbar:

- JAVA als Basisplattform ermöglicht die Verwendung verschiedenster stationärer und mobiler Endgeräte.
- Der User Interface Agent stellt relativ geringe Anforderungen an die Ressourcen des Endgerätes.
- Die Realisierung der Präsentationslogik in Form von multimodalen Rendermodulen bietet ein hohes Maß an Flexibilität.
- Das System ist aufgrund des Frameworks für die Rendermodule relativ einfach zu erweitern, ohne dass in das Kernsystem eingegriffen werden muss.
- Der funktionale Kern ist in den mobilen Agenten realisiert. Daher ist der Offline-Modus realisierbar.

Kriterien \ Systeme	MUSA	.NET	EMBASSI	SmartKom	MAP-UI-Agent
			Plattformen (stationär)		
WindowsXP	✓	✓	✓	✓	✓
MacOS	✓		✓		✓
Linux/UNIX	✓		✓	✓	✓
			Plattformen (mobil)		
PalmOS	✓		✓		✓
Windows Mobile	✓	✓	✓	✓	✓
Symbian	✓		✓	✓	✓
			Adaptionsverfahren		
Sequentialisierung	✓	✓			✓
Strukturerhaltung	✓			✓	✓
Verbessert Navigation	✓		✓	✓	✓
Unterstützt Assistenz			✓	✓	✓
			Modularität		
Einfache Anpassbarkeit	✓	✓	✓		✓
Einfache Erweiterbarkeit	✓	✓	✓		✓
			Betriebsarten		
Remote Evaluation (REV)	✓	✓		✓	✓
Code On Demand (COD)	✓	✓			✓
Mobile Agenten					✓
Offline					✓
			Mobilität		
Gerätewechsel möglich			✓	✓	✓
Erhaltung des Dialogzustands			✓	✓	✓
Berücksichtigung Gerätefähigkeiten			✓	✓	✓

Tabelle 21: Die Systeme im Vergleich

8 Zusammenfassung und Ausblick

Dieses Kapitel dient der Zusammenfassung der durchgeführten Arbeiten und erzielten Ergebnisse. Dabei werden die Ergebnisse insbesondere hinsichtlich der Fragestellungen von grafisch-interaktiven Systemen und mobiler Mensch-Maschine Interaktion diskutiert. Die Anwendungsgebiete der Thematik und Konzepte dieser Arbeit werden, ausgehend vom Anwendungsbeispiel, erweitert. Schließlich werden im Rahmen eines Ausblicks weiterführende Arbeiten benannt.

Die vorliegende Arbeit hat ein Konzept zur Bereitstellung eines adaptiven und mobilen User Interface Management Systems für den Einsatz mit Softwareagenten aufgestellt. Dieses Konzept ist motiviert durch den zunehmenden Verbreitungsgrad von mobilen und stationären EDV-Geräten, welche die Nutzung der Endgeräte als gemeinsame Plattform für Anwendungsprogramme nahe legt. Die sich ergebenden Potenziale für neue mobile Anwendungen müssen identifiziert und geprüft werden, um die Konsequenzen für das Arbeiten und Wirtschaften schon im Vorfeld dieser sich anbahnenden Innovation zu erforschen. Die Erforschung dieser Technologien bedingt die Modellierung geeigneter Architekturen und die Realisierung von Prototypen, anhand derer die Validierung möglich ist.

Eine zentrale Rolle in künftigen Szenarien spielt die Mensch-Maschine Interaktion. Werden die besonderen Anforderungen für die mobile Interaktion nicht berücksichtigt, die sich durch die inhomogene Zusammensetzung der Endgeräte ergeben, ist die Umsetzung von mobilen Anwendungen nicht möglich. Die vorliegende Arbeit leistet genau hier einen entscheidenden Beitrag. Es wird der Frage nachgegangen, wie für Softwareagenten die Interaktionsmöglichkeiten mobiler Endgeräte zugänglich gemacht werden können.

Zur näheren Betrachtung der mobilen Mensch-Maschine Interaktion wurde mittels eines repräsentativen Szenarios an den Sachverhalt herangegangen. Die Analyse des Szenarios ergab funktionale Anforderungen an die mobile Mensch-Maschine Interaktion nach vier unterschiedlichen Aspekten: anwendungsorientierte Anforderungen, anwenderorientierte Anforderungen, entwicklerorientierte und allgemeine Anforderungen. Die Anforderungen sind am Ende des Kapitels zu einem Kriterienkatalog zusammengefasst. Die Kriterien an mobile interaktive Anwendungen dienen einerseits der Identifikation benötigter Komponenten eines UIA und der Bewertung von Lösungswegen.

Es stellte sich aufgrund der szenarischen Analyse heraus, dass ein geeignetes System folgende Merkmale aufweisen sollte:

- *Die Abstraktion der Interaktionsdaten von den Geräteschnittstellen:* Damit ist es möglich eine Benutzungsoberfläche einmal abstrakt zu definieren und dann immer wieder zu verwenden.

- Mobile und stationäre Endgeräte können *Kombinationen* mehrerer Klassen von Interaktionsschnittstellen gleichzeitig, also *multimodal* benutzen.
- Ein Dialog kann *kontinuierlich* über mehrere Geräte hinweg bearbeitet werden.

Um eine gemeinsame Betrachtungsweise zu erreichen und funktionale Komponenten zu einer Lösung anordnen zu können wurde aus den Anforderungen und bestehenden Modellen ein Konzept entworfen das die mobile Mensch-Maschine Interaktion im geforderten Sinne ermöglicht. Das Konzept wurde hinsichtlich des Anforderungskatalogs geprüft. Offenbar erfüllt das Konzept alle daran gestellten Anforderungen.

Zum Nachweis der Realisierbarkeit ist eine prototypische Implementation erfolgt, siehe Kapitel 6. Der Prototyp geht auf das vorgestellte Lösungskonzept aus Kapitel 5 zurück. Der Prototyp kann die entworfene abstrakte Interaktionsbeschreibungssprache ausführen und Präsentationen mit verschiedenen ausgeprägten Endgeräten vornehmen.

Um die Güte des Ansatzes, seine Wirkungsweise und Qualität im Hinblick auf die in Kapitel 2 formulierten Anforderungen validieren zu können, wurde das Konzept im MAP Projekt eingesetzt. Die Ergebnisse der im Projektrahmen unternommenen Benutzertests wurden in Kapitel 6 ebenfalls dargestellt. Abschließend wurde die Lösung mit dem in Kapitel 4 erarbeiteten Stand der Technik verglichen. Damit konnte die Plausibilität und die Überlegenheit des Systemansatzes gegenüber verfügbaren Lösungen für den spezifizierten Anwendungsfall nachgewiesen werden.

Der Prototyp erfüllt alle an ihn gestellten Anforderungen in hervorragender Weise. Darüber hinaus ist er so flexibel, das er noch für weitere Anwendungsszenarien eingesetzt werden kann [Blec04]. Die Bedienbarkeit kann durch Einbeziehung von Domäneninformation in die Navigation gesteigert werden. Dadurch würde sich der Aufwand bei der Interaktion noch einmal verringern. Ein entsprechender Ansatz dafür wurde in [BS02] vorgestellt.

Die für das umgesetzte System entwickelten Rendermodule stellen eine repräsentative Teilmenge möglicher Anwendungsfälle dar und reichen von direkt manipulativen WIMP-Interfaces bis hin zu Assistenz- und Delegationsunterstützenden Systemen, die anthropomorphe, virtuelle Avatare zur Benutzerinteraktion nutzen. Ohne Probleme ließe sich noch die Anzahl der unterstützten mobilen Geräte erhöhen, durch Programmierung zusätzlicher Rendermodule. Solche Geräte können Teile bisher nicht betrachteter Szenarien sein, wie etwa öffentliche Fahrkartenautomaten, etc.

Die Benutzerdialoge können problemlos mit Standard XML Editoren erstellt und mit den Präsentationskomponenten des Prototypen dargestellt werden. In einer zukünftigen Anwendung ist es denkbar die Erstellung der Dialoge noch weiter zu vereinfachen, etwa durch die Bereitstellung einer spezialisierten Autorenumgebung.

Das in dieser Arbeit vorgestellte Konzept kann, neben dem konkreten Projekt und den Entwicklungen, in genereller Hinsicht auf die Entwicklung von Graphisch Interaktiven Systemen und der Informatik gewertet werden.

So ist eine Erweiterung des Gebietes um adaptive, mobile Strukturen notwendig, die nicht nur eine Assistenz und Delegation sowie eine Mensch-ähnliche Kommunikation erlauben. Vielmehr

soll dem Benutzer der allgegenwärtige Umgang mit der Maschine durch mobile Techniken und damit die Nutzung verschiedenster, räumlich getrennter Informationsräume, tatsächlich ermöglicht werden. Dies impliziert, dass der Übergang von Einem zum anderen Informationsraum schließlich verwischt. Hierzu ist weitere Arbeit bezüglich der mobilen graphisch-interaktiven Präsentation und Interaktion mit Menschen notwendig, dies sowohl im Bezug zum *Perceptual Computing* als auch zu *Intelligenten Umgebungen*, welche sich als Komponenten der Computer Graphik gerade etablieren.

Der Ausblick zeigt, dass die vorliegende Arbeit einen Startpunkt für weiterführende Arbeiten auf dem Gebiet der mobilen Mensch-Maschine Interaktion bietet. Die in dieser Arbeit vorgestellte XML Struktur bildet dabei einen generellen Rahmen für weiterführende Arbeiten.

Die Verknüpfung von Mobilität, Interaktion und natürlichen Dialogen bleibt bei weiterführenden Arbeiten erhalten – dies schon Aufgrund der komplexen Mechanismen, wie sie in der natürlichen Kommunikation wirken, welche eine Zielvorgabe der Interaktiven Computergraphik ist.

ANHANG

LITERATURVERZEICHNIS

[AMR00] Azevedo, P., Merrick, R., Roberts, D. *OVID to AUIML – User-Oriented Interface Modelling.* TUPIS'2000 Workshop Proceedings, York, 2./3, Oktober 2000.

[AR00] André, E., Rist, T. *Presenting Through Performing: On the Use of Multiple Lifelike Characters in Knowledge-Based Presentation Systems.* Proc. Of the Second International Conference on Intelligent User Interfaces (IUI 2000), pp. 1-8, 2000.

[BB03] Blechschmitt, E., Berner U. *A Dialog based User-Interface for Different Devices and Modalities.* International Workshop on Mobile Computing, IMC 2003, Rostock, Germany, June 17-18, 2003.

[BB94] Bharat, K. and Brown, M. H. *Building Distributed Multi-User Applications By Direct Manipulation.* Proc. ACM Symposium on User Interfaces Software and Technology, Marina Del Rey, CA, s. 71-82, Nov 1994.

[BC95] K. Bharat and L. Cardelli. *Migratory Applications*. In Proceedings of the Eighth ACM Symposium on User Interface Software and Technology, s. 133-142, Nov 1995.

[BE00] Bylund, M., Espinoza, F. *sView—personal service interaction*. In: Proceedings of the 5th international conference on the practical application of intelligent agents and multi-agent technology (PAAM 2000), Manchester, UK, April 2000.

[BFLM+92] Bass, L., Franeuf, R., Little, R., Mayer, N., Pellegrino, B., Reed, S., Seacord, R., Sheppard, S. and Szczur, M. *A Metamodel for the Runtime Architecture of an Interactive System.* The UIMS Tool Developers Workshop, SIGCHI Bulletin, 24(1), pp. 32-37, January 1992.

[BBGM+01] Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., Zukowski, D. *Challenges: An application Model for Pervasive Computing*. ACM Press New York, NY, USA, 2001.

[BLB01] Blechschmitt, E., Laube, M., Berner, U. *ZE 2.5.1 a Integration für „User Interface Agent"-Demonstrator.* Technical Report, Fraunhofer Institut Graphische Datenverarbeitung, Darmstadt, 22. Februar 2002.

[Blec04] Blechschmitt, E. *A SYSTEM THAT PROVIDES ADAPTIVE AND CONTINEOUS INTERACTION FOR MOBILE E-BUSINESS*. E-Commerce 2004, IADIS International Conference, Lisbon, Portugal, December 14-16, 2004.

[Blec99] Blechschmitt, E., Hockauf, B., Sahm, J., Dohrmann, C., Brümmer, A. *OfficePlus – Felderprobung des multimedialen Arbeitsplatzes der Zukunft*, Zwischenbericht 1999, Fraunhofer-IGD, 99i046-FIGD, 1999.

[Bolt80] Bolt, R. A. *Put that there: Voice and Gesture at the graphics interface*. ACM Computer Graphics 14, 3 (1980), 262-270, 1980.

[Brau01] Braun, N. *ZE 2.4.6a: Gestalterische Konzeption eines API für die Ansteuerung der Konversationsschnittstelle.* Technical Report. Fraunhofer Institut Graphische Datenverarbeitung. Darmstadt. 8. Oktober 2001.

[Brau02] Braun, N. *Symbolic Conversation Modeling Used As Abstract Part Of The User Interface*. WSCG 2002 Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, Plzen, Czech. Republic, 2002.

[Brit01] Britton, K., Case, R., Citron, A., Floyed, R., Li, Y., Seekamp, C., Topol, B., Tracey, K. *Transcoding: Extending e-business to new environments*. IBM Systems Journal 40, pp. 153-178, 2001.

[BS01] Blechschmitt, E., Sahm, J. *An agent-based system's architecture to describe workflows in the office*. E-Business and e-Work Conference (e2001), Venice, Italy, October 2001.

[BS02] Blechschmitt, E., Strödecke, C. *An Architecture to provide adaptive, synchronized and multimodal Human Computer Interaction*. Proceedings of the ACM Multimedia 2002 ,Juan les Pins, France, December 2002.

[BS03] Blechschmitt, E., Strödecke, C. *Non Hierarchical Mergeable Dialogs.* 10th International Conference on Human – Computer Interaction, HCII2003, Crete, Greece, June 22-27, 2003.

[Busc96] Buschmann, F., Meunier, R., Rohnert H., Sommerlad, P., Stal, M. *Pattern-Oriented Software Architecture – A System Of Patterns.* John Wiley & Sons Ltd. ISBN: 0-471-95869-7, September 1996.

[CCN97] Calvary, G., Coutaz, J., and Nigay, L. *From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW,* In CHI, pages 242-249, 1997.

[Cout87] Coutaz, J. *PAC, an Object Oriented Model for Dialog Design*. Human – Computer Interaction – INTERACT'87, Conference Proceedings, S. 431-436, IFIP, 1987.

[Cout95] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R. *Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties*. In Proceedings of INTERACT'95, Lillehammer, Norway, June 1995.

[Dest05] Statistisches Bundesamt, *Ausstattung privater Haushalte mit Informations- und Kommunikationstechnik in Deutschland*, Ergebnisse der laufenden Wirtschaftsrechnungen 2002-2004, verfügbar unter http://www.destatis.de/basis/d/evs/budtab2.php, (letzter Zugriff am 17.5.2005)

[Dist00] Distel, R. *Graphentheorie.* Springer-Verlag, Heidelberg, ISBN 3-540-67656-2, 2000.

[Duca01] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J. & Burgelman, J-C. *The European Union report, Scenarios for Ambient Intelligence in 2010*. ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf, (letzter Zugriff am 20.06. 2005), 2001.

[EMB02] *Elektronische Multimediale Bedien- und Service-Assistenz.* White Paper (V 1.1, August 2002), Verfügbar unter http://www.embassi.de (20.06. 2005).

[Enca03] Encarnação, J. L. *Die Integration des Menschen in intelligente IT-Umgebungen.* In: Kunststück Innovation, S. 36-45. Berlin. Heidelberg. New York: Springer Verlag, 2003. ISBN: 3-540-43987-0, 2003.

[Enca99] Encarnação, J. L. *Schlüsseltechnologien für intelligente Produkte und Dienstleistungen*. Kapitel Zukunftsweisende F&E-Trends und Anwendungen für neue Medien in der Informations- und Kommunikationstechnologie, Seiten 15–41. VDI F+E Forum, Mannheim, 1999.

[FIPA] *The Foundation For Intelligent Physical Agents*. Internetseiten der FIPA Organisation, http://www.fipa.org, (letzter Zugriff am 20.06.2005).

[FIPAa] *FIPA Request Interaction Protocol Specification*. Foundation for Intelligent Physical Agents, 2000. http://www.fipa.org/specs/fipa00023, (letzter Zugriff am 20.06.2005).

[FMP02] Fischmeister, S., Menkhaus, G., und Pree, W. *MUSA-Shadow: Concepts, Implementation, and Sample Applications; A Location-Based Service Supporting Multiple Devices.* In Proceedings of TOOLS Pacific, 71-79, Sydney, Australia, February 2002.

[FT02] Fehr, M., TaQuang, T. *Erstellung Testpläne für Feldtest*. MAP Interner Bericht Nr. map-capc007-020521-v01, 2002.

[FVFH95] Foley, J., D., van Dam, A., Feiner, S., K., Hughes, J., F. *Computer Graphics – Principles and Practice*, Addison-Wesley Pub Co. 2nd edition, ISBN: 0201848406, 1995.

[GGPY89] Gelsinger, P., Gargini, P. A., Parker, G. H., Yu, A. Y. C. *Microprocessors Circa 2000. The microprocessor of the next century is going to be faster and more powerful…than a mainframe*. IEEE Spectrum, pp. 43-47, October 1989.

[GHJV98] E. Gamma, R. Helm, R. Johnson, J. Vlissidis, *Entwurfsmuster: Elemente wiederverwendbarer objekt-orientierter Software*, 2. Auflage, Addison-Wesley, 1998.

[Gree85] Green, M. *Report on Dialogue Specifcation Tools*. In Günther E. Pfaff (Ed.) User Interface Management Systems. Springer Verlag, 1985.

[Han00] Handrick, S. *Arbeitswissenschaftliche Kriterien zu MAP*. Zwischenergebnis map-tudr002. MAP Konsortium, 2000.

[HBCC+02] Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong & Verplank. *ACM SIGCHI Curricula for Human-Computer Interaction.* http://www.acm.org/sigchi/cdg/cdg2.html. (letzter Zugriff am 20.06.2005).

[Heis02] *Eine Milliarde Handys weltweit*. Meldung des Online News-Tickers des Heise Verlags. http://www.heise.de/newsticker/meldung/24732, (letzter Zugriff am 20.06.2005), Februar 2002.

[Heis03] *E-Plus bringt Handy-PDA mit GPRS-Flatrate.* http://www.heise.de/newsticker/meldung/41229, (letzter Zugriff am 20.06.2005), Oktober 2003.

[HK02] Heider Th., Kirste T. *Architecture considerations for interoperable multi-modal assistant systems* PreProceedings of the 9th International Workshop on Design, Specification, and Verification of Interactive Systems; DSV-IS 2002, 6, 2002, Rostock, S.403 – 417, 2002.

[HKMN+03] Herzog, G., Kirchmann, H., Merten, S., Ndiaye, A., Poller, P., Becker, T. *MULTIPLATFORM Testbed: An Integration Platform for Multimodal Dialog Systems*. HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems (SEALTS), Edmonton (Kanada), 31.05.2003.

[IBM96] *The IBM aglets workbench.* Aglets Specification 1.1 Draft, IBM Corp. 1996. http://www.trl.ibm.com/aglets, (letzter Zugriff am 20.06.2005).

[IEEE99] *IEEE 802.11b-1999 (R2003) Supplement to 802.11-1999,Wireless LAN MAC and PHY specifications: Higher speed Physical Layer (PHY) extension in the 2.4 GHz band*, http://standards.ieee.org/catalog/olis/lanman.html, (letzter Zugriff am 20.06.2005).

[ISO14977] *ISO/IEC 14977 : 1996(E) – ISO Standard zu EBNF*. (final draft version SC22/N2249), http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf, (letzter Zugriff am 20.6.2005).

[ISO9241] *DIN EN ISO 9241 Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten.* Europäische Normen, 1996.

[J2ME] *Java 2 Platform, Micro Edition (J2ME)*, Online Dokumentation, http://java.sun.com/j2me/index.jsp, (letzter Zugriff am 20.06.2005).

[JADE] *Projekt-Homepage der JADE-Agentenplattform.* http://sharon.cselt.it/projects/jade, (letzter Zugriff 20.06.2005).

[JAVA] *Java™ 2 SDK Dokumentation.* Verfügbar als PDF and PS Version unter http://java.sun.com/j2se/1.3/download-pdf-ps.html, (letzter Zugriff 20.06.2005).

[KB96] Kazman, R., Bass, L. *Software Architectures for Human-Computer Interaction: Analysis and Construction.* Submitted to ACM Transactions on Human-Computer Interaction, 1996.

[Kirst02] Kirste, T., u.A. *Die Zukunft interaktiver Systeme: Herausforderungen an die Usability-Forschung.* It+ti 1/2002, S. 40-48, 2002.

[KP01] MAP Konsortium and MAP Projektbüro. *Multimedia-Arbeitsplatz der Zukunft: Managementzusammenfassung*. Technical Report, Fraunhofer Institut Graphische Datenverarbeitung, Darmstadt, 2001.

[KP88] Krasner, G. E. and Pope S. T. *A cookbook for using the model-viewcontroller user interface paradigm in smalltalk-80*. Journal of Object-Oriented Programming, 1(3). Seiten 26-49, August/September 1988.

[Kuhl01] Kuhlmann, H. *Multimedia Workplace of the Future*. Office**,** e-Business and e-Work Conference (e2001), Venice, Italy, October 2001.

[LEAP] *Lightweight Extensible Agent Platform.* Internetseiten des LEAP-Projekts, 5th Framework der IST, Project Contract: IST-1999-10211, http://leap.crm-paris.com, (letzter Zugriff am 20.06.2005).

[LLCR02] Luyten, K., van Laerhoven, T., Coninx, K. und van Reeth, F. *Specifying User Interfaces for Runtime Modal Independent Migration*. Paper presented at CADUI 2002, Fourth International Conference on Computer-Aided Design of User Interfaces, Université de Valenciennes, France, May 15-17, 2002.

[Maes94] Maes, P. *Agents that reduce Work and Information Overload*. Communication of the ACM, Vol. 37, Nr. 7, S. 31-40, July 1994.

[Mang02] Mangina, E. *Review of Software Products for Multi-Agent Systems*. In: AgentLink, software report, 2002.

[MAP21] *Multimedia Arbeitsplatz der Zukunft*. Offizielle Internetseiten des MAP-Projekts, http://www.map21.de, (letzter Zugriff am 20.6.2005)

[MF02] Menkhaus, G., Fischmeister, S. *Adaptation for Device Independent Authoring.* In Developing User Interfaces with XML: Advances on User Interface Description Languages Workshop at Advanced Visual Interface (AVI 2004), Gallipoli, Italy, 2004.

[Micro02a] Microsoft .NET. *Defining the Basic Elements of .NET*. April 2002. http://www.microsoft.com/net/basics/whatis.asp, (letzter Zugriff am 20.06. 2005), April 2002.

[Micro02b] Microsoft .NET. *Smart Device and .NET*. January 2002. http://www.microsoft.com/net/products/devices.asp, (letzter Zugriff am 20.06.2005), January 2002.

[Moor65] Moore, G. E. *Cramming more components onto integrated circuits.* Electronics, Volume 38, Number 8, 19. April 1965.

[Mozi00] *XPToolkit Project.* Mozilla.org 2000, http://www.mozilla.org/xpfe, (letzter Zugriff am 20.06.2005).

[MR92] Myers B., A. and Rosson, M., B. *Survey on User Interface Programming.* Proceedings of SIGCHI'92: Human Factors in Computing Systems, May 3-7, 1992.

[Myer93] Myers, B., A. *Why are Human-Computer Interfaces Difficult to Design and Implement?* Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-93-183. July 1993.

[NB03] Nylander, S., Bylund, M. *The Ubiquitous Interactor: Universal Access to Mobile Services.* 10th International Conference on Human-Computer Interaction, HCII2003, Crete, Greece, June 22-27, 2003.

[NBW05] Nylander, S., Bylund, M., Waern, A. *Ubiquitous service access through adapted user interfaces on multiple devices.* Personal and Ubiquitous Computing, Volume 9, Issue 3, Pages 123 – 133, May 2005.

[NC95] Nigay, L., and Coutaz, J. A Generic Platform for Addressing the Multimodal Challenge. Proceedings CHI'95, (Denver CO, May 1995), May 1995.

[Nech91] Neches, R. et al. "*Enabling Technology for Knowledge Sharing*". In *AI Magazine*, Volume 12, Number 3, pp. 36-56, 1991.

[Norm86] Norman, D. A., Draper, S. W., *User Centered System Design. New Perspectives on Human-Computer Interaction*. Journal Dept. Lawrence Erlbaum Associates, Inc. 365 Broadway, Hillsdale, NJ 07642. ISBN 0-89859-872-9, 1986.

[Norm94] Norman, D., *How might people interact with agents.* Communications of the ACM, 7 (1994), 68, 1994.

[Nye90] Nye, A., O´Reilly, T. *X Toolkit Intrinsics Programming Manual – The definitive Guide to the X Window System (Vol. IV)*. O´Reilly & Associates, Sebastopol, CA, 1990.

[OASIS] Organization for the Advancement of Structured Information Systems. http://www.oasis-open.org, (letzter Zugriff am 20.06.2005).

[OMA01] *WAP Architecture - Version 12-July-2001.* Spezifikation des WAP, Internetseiten der Open Mobile Alliance (OMA) unter http://www.openmobilealliance.org, (letzter Zugriff am 20.06.2005).

[Ovia02] Oviatt, S. *Multimodal Interfaces.* In: The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications. Lawrence Erlbaum Assoc. S. 286 – 304, ISBN: 0805844686, 2002.

[Ovia99] Oviatt, S. *Ten Myths of Multimodal Interaction. Moving from traditional interfaces toward interfaces offering users greater expressive power, naturalness, and portability.* Communications oft the ACM, Vol. 42, No. 11, November 1999.

[Pinsd02] Pinsdorf, U., Peters, J., Hoffmann, M. & Gupta, P., *Context-aware Services based on Secure Mobile Agents.* Proceedings of 10th International Conference on Software, Telecommunications & Computer Networks (SoftCOM 2002), pages 366-370, IEEE Communication Society, Ministry of Science and Technology Republic of Croatia and University of Split, R. Boskovica, HR-21000 Split, Croatia, October 2002.

[Preec94] Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T. *Human-Computer Interaction*, Addison-Wesley. 1994. ISBN: 0201627698, 1994.

[Puer02] Puerta, A., Eisenstein, J., *XIML: A Common Representation for Interaction Data.* IUI'02, January 13-16, San Francisco, California, USA, 2002.

[Puer97] Puerta, A.R. *A Model-Based Interface Development Environment*, In IEEE Software, pp. 40-47, 1997.

[Puer99] Puerta A. and Eisenstein, J. *Towards a General Computational Framework for Model-Based Interface Development Systems.* In: Knowledge-Based Systems, Vol. 12, pp. 433-442, 1999.

[Rath95] Rath, R. *Über das A und O des Telefonierens.* Lili-Zeitschrift für Literatur-wissenschaft und Linguistik. 25, 9-35, 1995.

[RE03] Richter, K., Enge, M. *Multi-modal Framework to Support Users with Special Needs in Interaction with Public Information Systems*. In: Chittaro, Luca (Ed.): Human-Computer Interaction with Mobile Devices and Services. Proceedings : Mobile HCI 2003. Berlin, Heidelberg, New York : Springer Verlag, pp. 286-301, 2003.

[Reich02] Reichswald, R. *Potenziale der mobilen Kooperation.* In Mobile Arbeitswelten - Soziale Gestaltung von „Electronic Mobility". Alcatel SEL Stiftung für Kommunikationsforschung/Forum für soziale Technikgestaltung (HRSG.). Talheimer Verlag. (Talheimer Sammlung kritisches Wissen; Bd. 35). ISBN-3-89376-087-3, 2002.

[RSWH+01] Rössler, H., Sienel, J., Wajda, W., Hoffman, J., Kostrzewa, M. *Multimodal Interaction for Mobile Environments.* Alcatel SEL AG Research and Innovation, 2001.

[Semoa] *Secure Mobile Agents Project.* Projektseiten, http://www.semoa.org, (letzter Zugriff 9.12.2004).

[SGBF01] Schmidt, A., Gellersen, H.-W., Beigl, M., Frick, O. *Entwicklung von WAP-Anwendungen*. Kommunikation in Verteilten Systemen (KiVS 2001). GI-Tagung, Hamburg, February 2001.

[SH01] Sandweg, N., Hermann, D. *Analyse der technischen Konzeption: Usability-anforderungen.* Zwischenergebnis 5.3.1. MAP Konsortium, 2001.

[Shne99] Shneiderman, B. *Designing the User Interface.* Pearson Addison Wesley. 3rd edition, ISBN: 0201694972, 1997.

[SM04] Schumann, H. und Müller, W. *Informationsvisualisierung: Methoden und Perspektiven*. In: it - Information Technology, 3/2004, S. 135-141, 2004.

[Smar03] *SmartKom.* Internetseiten des Leitprojekts, http://www.smartkom.org, (letzter Zugriff am 20.06.2005).

[SPN93] Szekely, P., Luo, P. and Neches, R. *Beyond Interface Builders: Model-Based Interface Tools*. In *Proc. of InterCHI'93*, ACM Press, New York, pp. 383-390, 1993.

[SVBL01] Sahm, J., Vollinga, J., Blechschmitt, E., Luckas, V. *Next Generation Geometry Data Exchange*. International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2001). L'Aquila, Italy, August 2001.

[SW76] Shannon, C. E., Weaver, W. *Mathematische Grundlagen der Informations-theorie*. R. Oldenbourg Verlag, 1976.

[Szek95] Szekely, P. et al. *Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach*. Engineering for Human-Computer Interaction, L.J. Bass and C. Unger (eds), Chapman & Hall, London, pp 120-150, 1995.

[UIML20] *User Interface Markup Language (UIML) Draft Specification. Document Version 17 January 2000. Language Version 2.0a*. Verfügbar unter http://www.uiml.org/specs (letzter Zugriff 20.06.2005).

[Vign98] Vigna, G. *Mobile Code Technologies, Paradigms, and Applications*. PhD Thesis, Politecnico di Milano/Italien, Februar 1998.

[Wahl03] Wahlster, W. *SmartKom: Symmetric Multimodality in an Adaptive and Reusable Dialogue Shell*, In: Krahl, R., Günther, D. (eds): Proceedings of the Human Computer Interaction Status Conference 2003, Berlin: DLR, p. 47-62, 3. Juni 2003.

[Wahr02] Wahrig, G., Wahrig-Burfeind, R. *Deutsches Wörterbuch.* Bertelsmann Lexikon Institut im Wissen Media Verlag. ISBN 3577100796, September 2002.

[WJ95a] Wooldrigde, M., Jennings, N. *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review 10 (2), 115-152, 1995.

[WJ95b] Wooldridge, M. & Jennings, N. *Intelligent Agents*, Lecture Notes in Artificial Intelligence 890, Heidelberg: Springer Verlag, 1995.

[WKL01] Weiss, M., Kuhlmann, H., Lotz, V. *Multimedia workspace of the future*. In Proceedings of the Statustagung der Leitprojekte Mensch-Technik-Interaktion, BMBF, 2001.

[WBBG+90] Wiecha, C., Bennett, W., Boies, S., Gould, J. and Greene, S. (1990) *ITS: a Tool for Rapidly Developing Interactive Applications*. ACM Transactions on Information Systems, 8**,** (3), 204-236, 1990.

[XFor03] *World Wide Web Consortium Publishes XForms 1.0 as a W3C Recommendation.* Pressemitteilung des W3C vom 14.10.2003. http://www.w3.org/2003/10/xforms-pressrelease.html.en, (letzter Zugriff 20.06. 2005).

[XIML] *Internet Seite des XIML-Forums*, http://www.ximl.org, (letzter Zugriff 20.06.2005).

[XML] *Extensible Markup Language (XML),* http://www.w3.org/XML, (letzter Zugriff 20.06.2005).

[ZVG02] Zimmermann, G., Vanderheiden, G. and Gilman, A. *Universal Remote Console Prototyping of an Emerging XML Based Alternate User Interface Access Standard*. Präsentation auf: Eleventh International World Wide Web Conference, 7-11 Mai 2002, Honolulu, Hawaii, 2002.

TABELLENVERZEICHNIS

ABBILDUNGSVERZEICHNIS

A. SCHRIFTENVERZEICHNIS

Dieser Anhang besteht aus einer Liste der Publikationen des Autors als auch der vom Autor betreuten studentischen Arbeiten.

A.1 Eigene Publikationen

In diesem Abschnitt sind die Publikationen des Autors aufgeführt.

- Eric Blechschmitt, *Adaptive Interface Development For The Mobile User*. Training, Education & Simulation International (TESI) 2005, MECC Maastricht, The Netherlands, 22–24 March 2005.
- Eric Blechschmitt, *A SYSTEM THAT PROVIDES ADAPTIVE AND CONTINEOUS INTERACTION FOR MOBILE E-BUSINESS*. e-Commerce 2004, IADIS International Conference, Lisbon, Portugal, December 14-16, 2004.
- Eric Blechschmitt, Christoph Strödecke. *Non Hierarchical Mergeable Dialogs.* 10th International Conference on Human - Computer Interaction, HCII2003, Crete, Greece, June 22-27, 2003.
- Eric Blechschmitt, Uwe Berner. *A Dialog based User-Interface for Different Devices and Modalities.* International Workshop on Mobile Computing, IMC 2003, Rostock, Germany, June 17-18, 2003.
- Eric Blechschmitt, Christoph Strödecke. *A Middleware For Conversational User Interfaces In Mobile Environments.* Proc. of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment, March 2003.
- Eric Blechschmitt, Christoph Strödecke. *The MAP-User Interface Agent*. Computer Graphik topics 6/2002, Vol. 14, 2002.
- Eric Blechschmitt, Christoph Strödecke. *An Architecture to provide adaptive, synchronized and multimodal Human Computer Interaction.* Association for Computing Machinery (ACM), Multimedia Conference 2002. Proceedings: MM 2002, pp. 287-290, New York, 2002.
- Eric Blechschmitt, Jörg Sahm. *An agent-based system's architecture to describe workflows in the office.* e-Business and e-Work Conference (e2001), Venice, Italy, October 2001.
- Jörg Sahm, Jens Vollinga, Eric Blechschmitt, Volker Luckas. *Next Generation Geometry Data Exchange.* International Conference on Advances in Infrastructure for

Electronic Business, Science, and Education on the Internet (SSGRR 2001), L'Aquila, Italy, August 2001.

- Eric Blechschmitt, Axel Brümmer, Volker Luckas, Brit Hockauf, Jörg Sahm. *OfficePlus. Abschlussbericht 2000*, In Fraunhofer-IGD Reportnr.: 01i005-FIGD, Darmstadt, 2001.
- Eric Blechschmitt, Brit Hockauf, Nicolai Maiorow, Jörg Sahm, Christoph Dohrmann. *OfficePlus. Jahresbericht 1999,* In Fraunhofer-IGD Reportnr.: 99i046-FIGD, Darmstadt 2000.
- Eric Blechschmitt, Christoph Dohrmann, Silke Höppner, Silke Karnop, Detlef Krömker, Wolfgang Müller, Jürgen Piesche, Kerstin Pipke, Jörg Sahm, Arno Schäfer; Ulrike Spierling. *Intelligente Werkzeuge für das Office 2005. Abschlussbericht.* In Fraunhofer-IGD Reportnr.: 99i018-FIGD, Darmstadt, 1999.
- Enkelmann, W., Balfanz, D., Blechschmitt, E., Dörner, R., Heidemann, M., Jasnoch, U., Krömker, D., Sobon, I., Heger, D., Kreutz, J., Kuhr, H. A., Nirschl, G., Steusloff, H., Danowski, K, Jung, U., Bernhard, J., Bretz, I., Klaus, P., Liberda, E., Pflaum, A., Prockl, G., Kraft, V., Kuchenbecker, M., Vastag, A., Zoche, P., Harmsen, D. M., Meißner, A., Linnemann, H. *Verkehrstelematik. Gemeinsame Studie - Version 4.* In Fraunhofer-IGD Reportnr.: 99i008-FIGD, Darmstadt, 1999.
- Eric Blechschmitt. *Antialiasing für LC-Displays*, In: Fraunhofer-IGD Reportnr.:98v002-FIGD, Darmstadt, 1998.
- Höppner, S., Krömker, D., Müller, W., Pipke, K., Spierling, U., Blechschmitt, E., Dohrmann, C., Karnop, S., Schäfer, A. *Intelligente Werkzeuge für das Office 2005 - Jahresbericht 1997.* In: Fraunhofer-IGD Reportnr.: 97i016-FIGD, Darmstadt, 1997.
- Eric Blechschmitt, Norbert Braun. *Studie Business TV.* In: Fraunhofer-IGD Reportnr.: 97v001-FIGD, Darmstadt, 1997.

A.2 Betreute studentische Arbeiten

In diesem Abschnitt sind die betreuten studentischen Arbeiten angeführt.

- Youssef Meddah. *Konzeption und Implementierung eines Protokollkonverters für die Kommunikation zwischen MBONE- und H.320-Videokonferenzsystemen*. Darmstadt, 1998.
- Mona Hanafi. *Markieren und Verfolgen von Objekten in MPEG Videodatenströmen.* Darmstadt, 1998.

B. LEBENSLAUF

Name:	Eric Blechschmitt	
Geboren:	10.1.1967	
Geboren in:	Darmstadt	
Ausbildung:	1974 - 1987	Schulbildung mit Abschluss Abitur, Gymnasiale Oberstufenschule Darmstadt
	1988 - 1996	Studium Diplom-Ingenieur Elektrotechnik mit Nebenfach Datentechnik an der Technischen Hochschule in Darmstadt
Berufspraxis:	1996 - 1997	Mitarbeiter beim Zentrum für Graphische Datenverarbeitung: Entwicklungsarbeiten an einem Videoserver
	1997 - 2004	Wissenschaftlicher Mitarbeiter beim Fraunhofer Institut für Graphische Datenverarbeitung in Darmstadt, Abt. Animation & Bildkommunikation.
	2000 - 2003	Schutzrechtsbeauftragter des INI-GraphicsNET